Beating Wallhacks using Deep Learning with Limited Resources

Junsik Hwang
Machine Learning Engineer @ Nexon Korea

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Topics

- problem definition: wallhack in FPS
- real problems & solutions
  - vs. limited data
  - vs. limited signal
  - vs. limited trust
- project output
- takeaways

Prerequisites

- What is Deep Learning
- How convolution layers work
- PyTorch (optional)

Problem Definition

Sudden Attack

- Developed by NexonGT released in 2005
- 15M+ Users & 260K+ MAU in South Korea
- #3 FPS in South Korea (PUBG > OW > SA)

Wallhacks in FPS

- ***See through walls***
- Ruins fair competition
- Less obvious than other hacks such as aim hacks
- Most commonly used

Existing Measures and Limitations

- Game software protection: first line of defence
  > abusers ***bypass*** the security check eventually

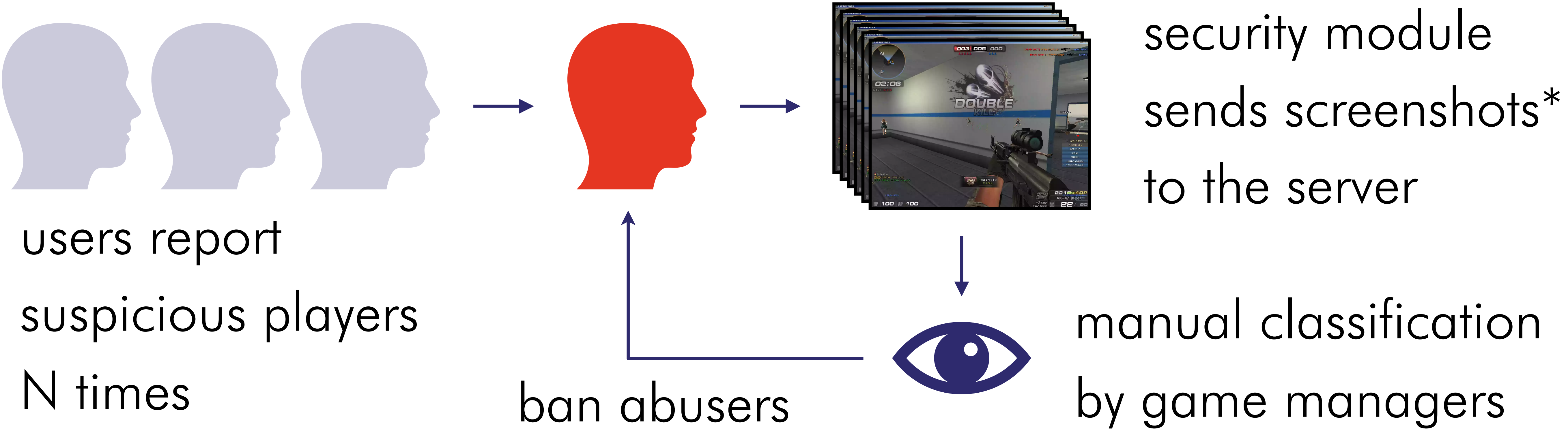
NEXON GAME SECURITY

- Server-side log analysis: outlier detection
  > ***hard how to tell*** "seeing through walls" with position coordinates and kill/death ratio

Live Bot Detection

Seeing is Detecting

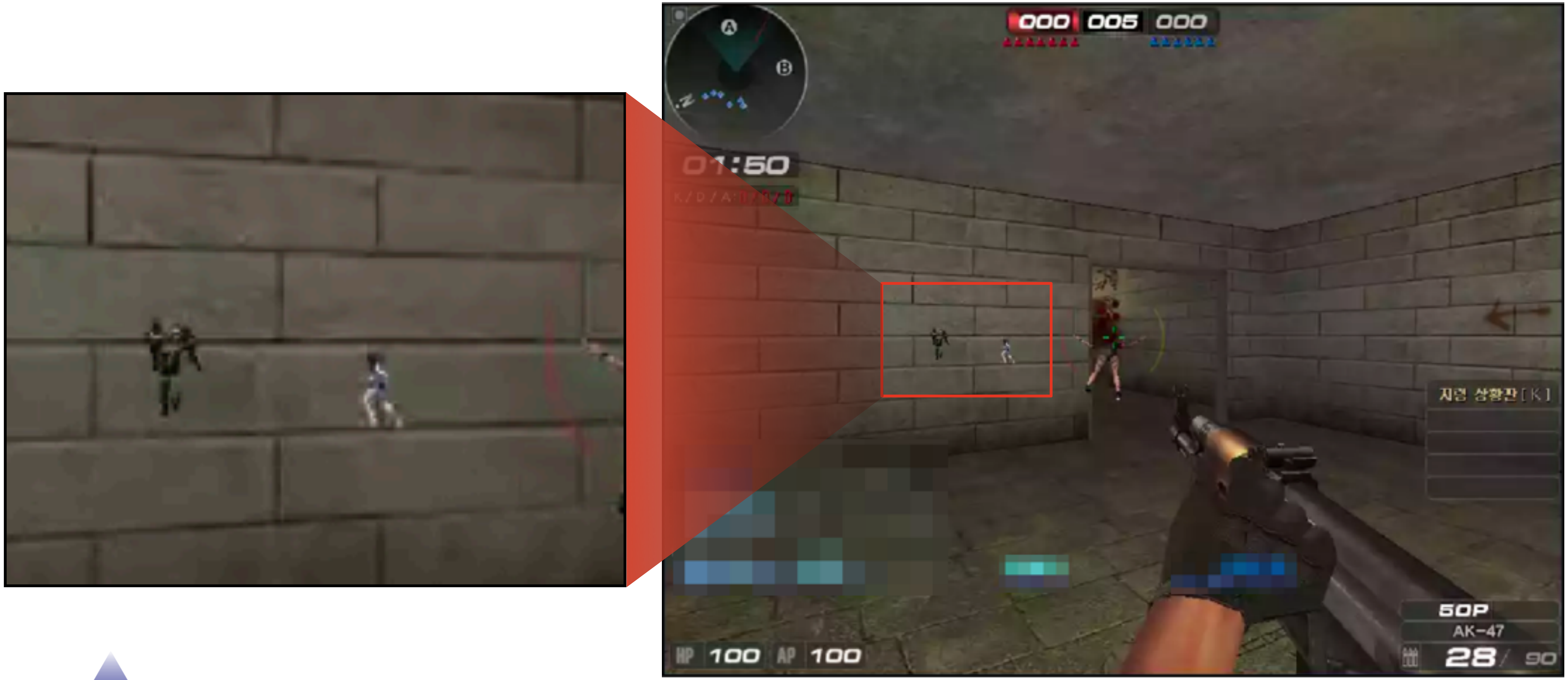
- The surest way to detect wallhacks is to ***see how they see.***



screenshots*: game screen only

Looks obvious to us, Human

Looks obvious to us, Human



Looks obvious to us, Human

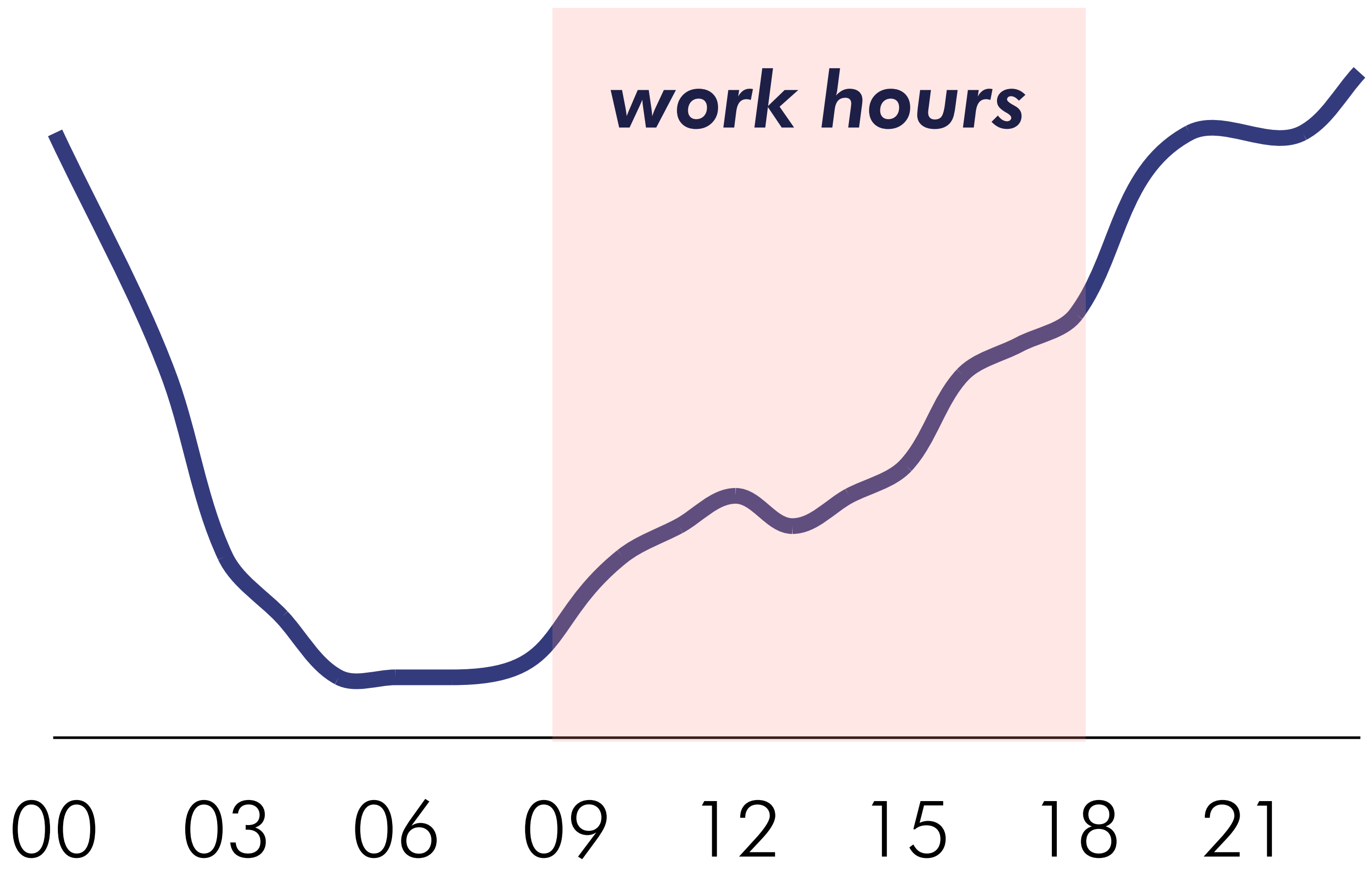
Looks obvious to us, Human

But Manual Inspection is Laborious

- Pros:
  - ***reliable*** / error-free

- Cons:
  - ***labour-intensive***
  - inspect 1,000 images/hour
  - gamers play when we rest

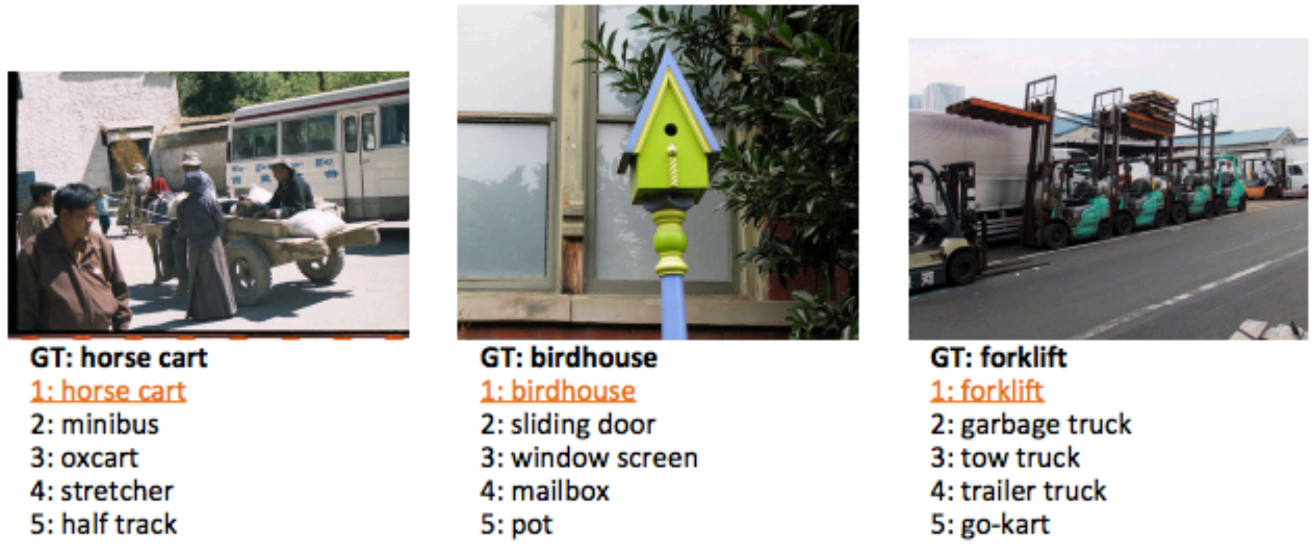
Number of Users per hour



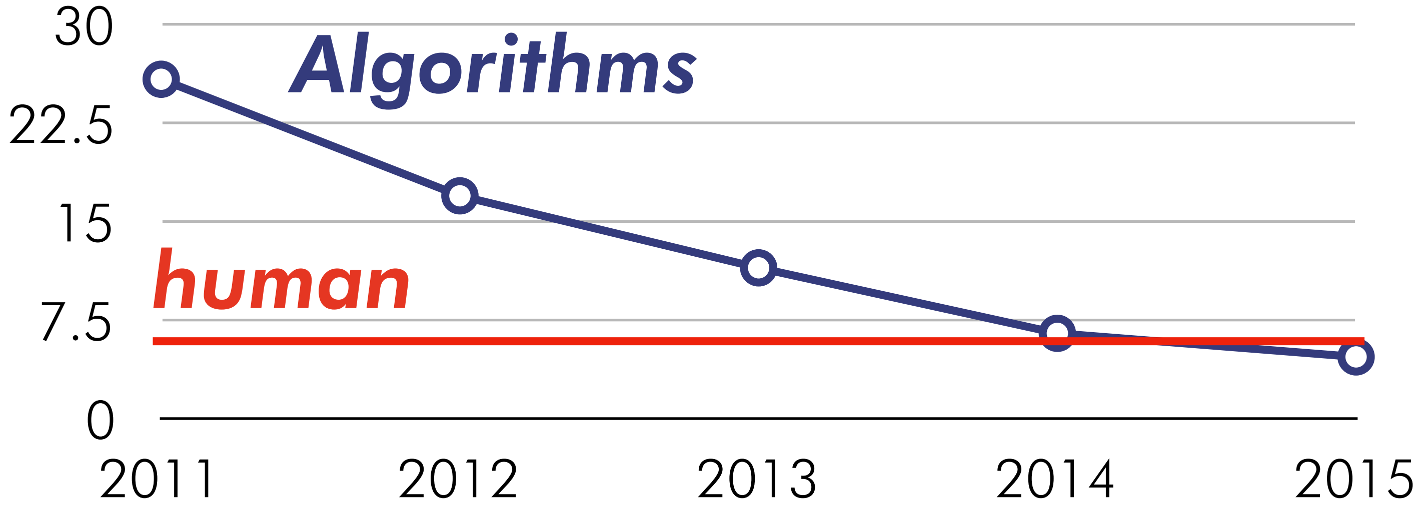
Deep Learning excels in Image Classification

- ImageNet classification task: classify 1,000 classes
- ResNet ***surpassed human*** baseline in 2015

ImageNet Task



top 5 error rates (%)



Beating Wallhacks using Human Labour

- Instead of going through images **one by one...**

users report
suspicious players
N times

security module
sends screenshots*
to the server

manual classification
by game managers

ban abusers

Beating Wallhacks using Deep Learning

- Let's ***automate the detection*** process with Deep Learning

users report
suspicious players
N times

security module
sends screenshots*
to the server

Automatic Ban
by Deep Neural Network

Beating Wallhacks using Deep Learning

- Let's ***automate the detection*** process with Deep Learning

users report suspicious players N times

security module sends screenshots* to the server

classification with ***Deep Neural Network***

ban abusers

And they lived *Happily* Ever After

Beating Wallhacks using Deep Learning

- Let's ***automate the detection*** process with Deep Learning

users report suspicious players N times

ban abusers

security module sends screenshots* to the server

classification with **Deep Neural Network**

And they lived

Happily

Ever After

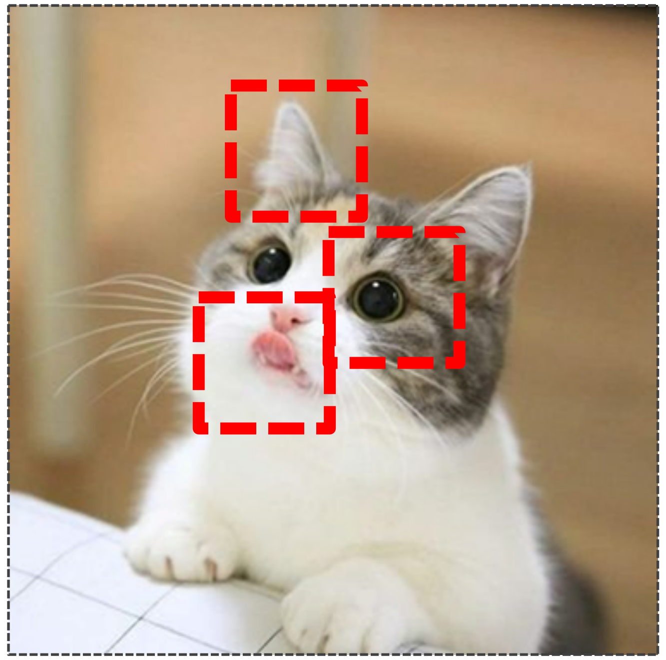
vs. Limited Data

Deep Learning requires Big Data

- Instead of handcrafted features, DL learn features from data
- Thus generally ***not suitable*** for small dataset

from numerous images

DL learns useful features

function () = "CAT" → 

And Big data requires Huge Investment

- Data acquisition / Preprocessing (ex. Labelling) are ***costly***
- We started with a mere 10,000 unlabelled images

MNIST

- 60,000 images
- fully labelled
- 28x28 (grayscale)

label = 5

Sudden Attack

- 10,000 images
- totally unlabelled & mixed
- 960x540 (RGB)

label = ?

Harder than I expected

- Low resolution images to save storage space
- Some images are very ***confusing***

Problem: 2,000 clean data to start with

- Prepared 2,000 wallhack / normal images (1,000 each)
- But is this ***big enough*** to run Deep Learning?

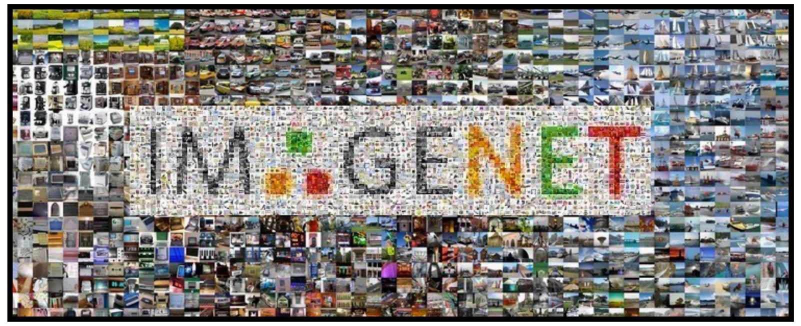
wallhack

normal

Solution: Transfer Learning

- ***Borrow feature extractor*** from a successfully trained model

1.2M images



▶

▶ 1,000 label classification

Feature extractor

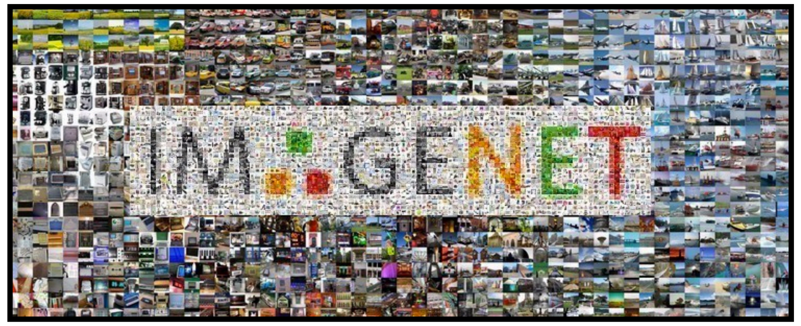
- Low level: lines & curves
- High level: shapes & texture

Classifier

Solution: Transfer Learning

- Don't train the whole model: ***fine-tune*** NN with small datasets

1.2M images



▶

▶ 1,000 label classification

↓ use ***pre-trained weights***

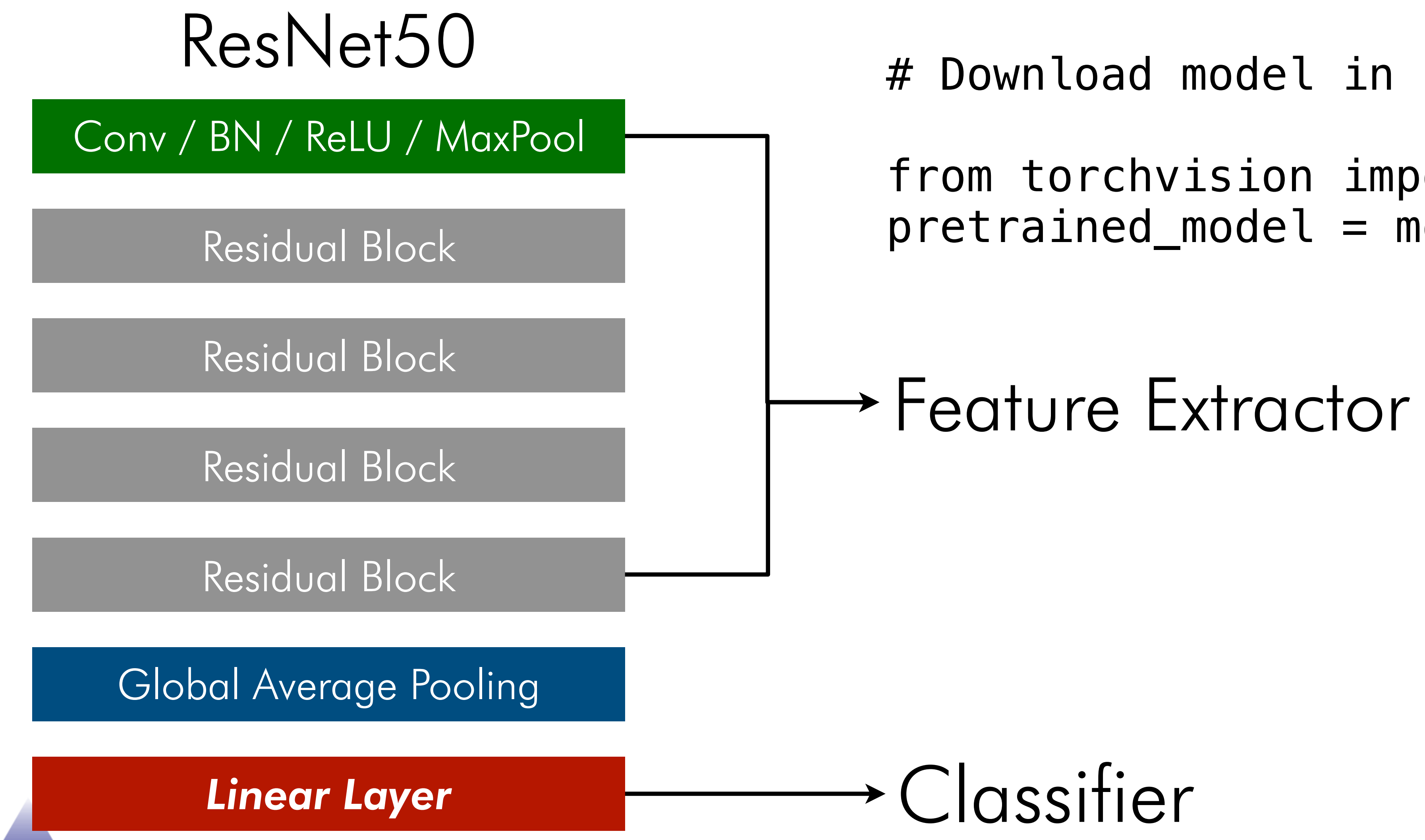
2,000 images

▶

▶ wallhack/normal binary classification

Pre-trained Layers + New Classifier

- Pre-trained models are available with PyTorch, TensorFlow, Keras

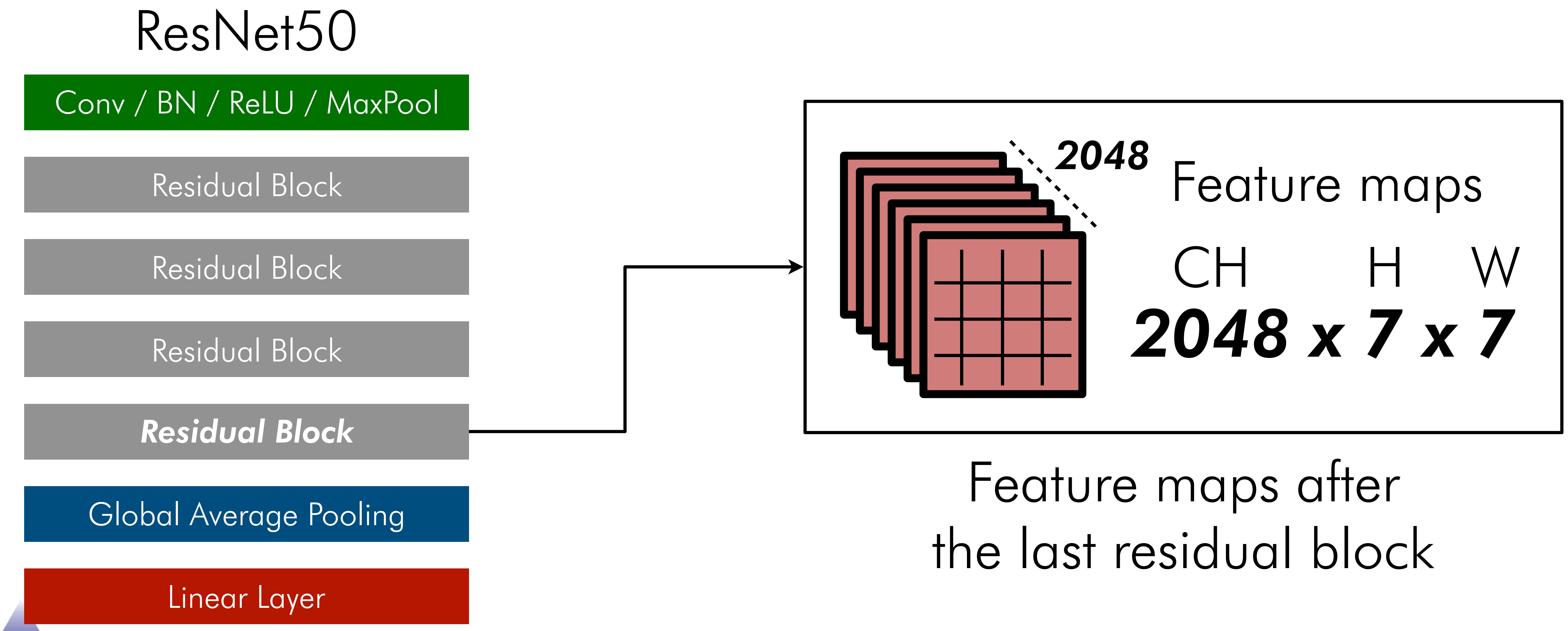



```
# Download model in PyTorch

from torchvision import models
pretrained_model = models.resnet50(pretrained=True)
```

Pre-trained Layers + New Classifier

- Pre-trained models are available with PyTorch, TensorFlow, Keras

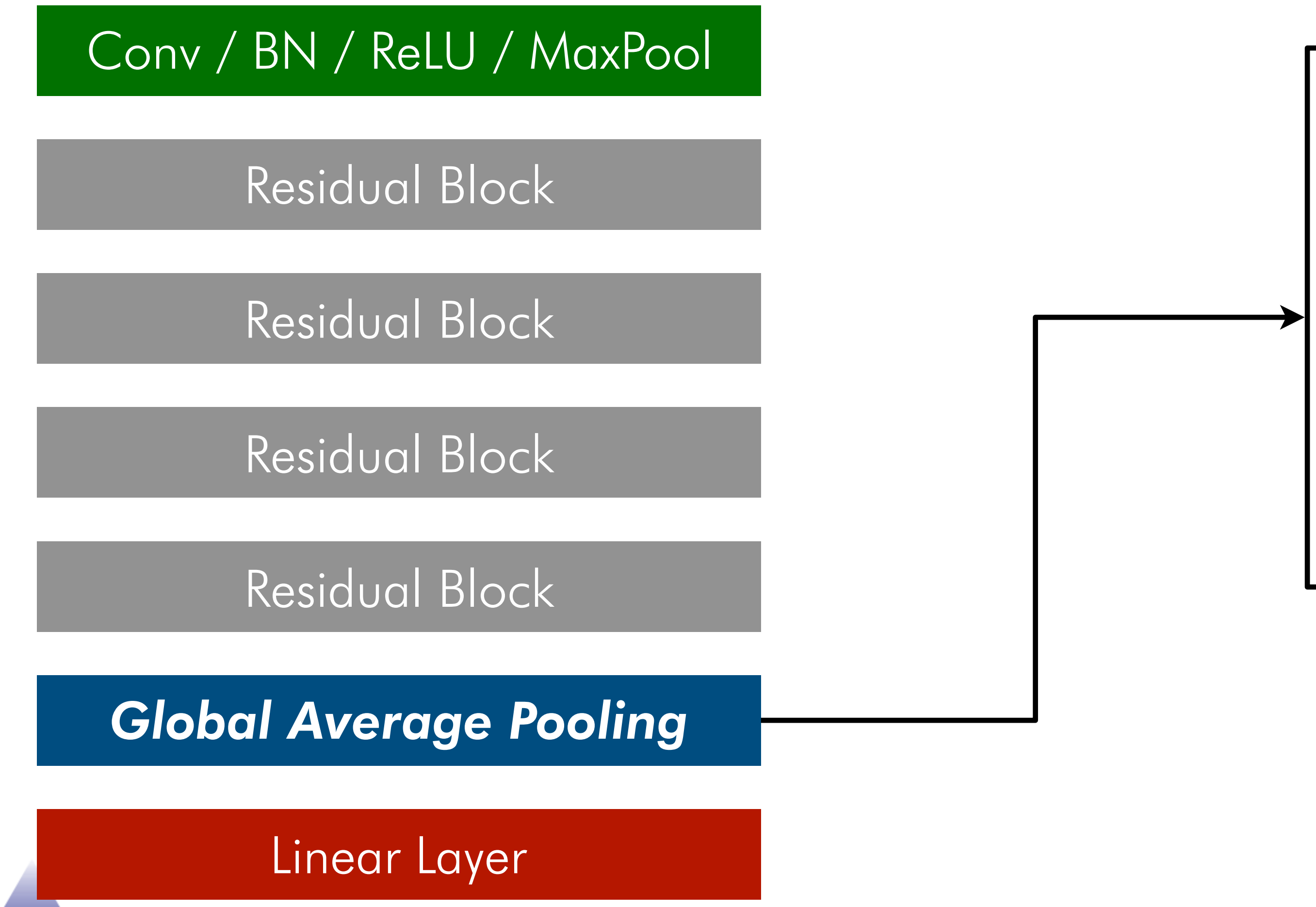


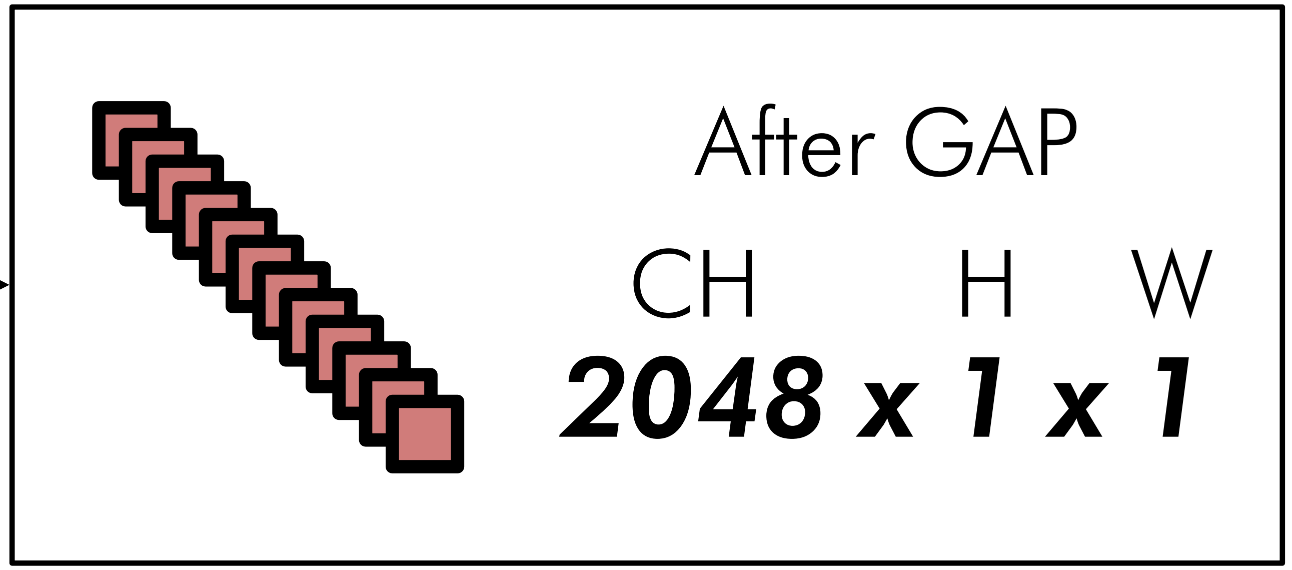
Feature maps after
the last residual block

Pre-trained Layers + New Classifier

- Pre-trained models are available with PyTorch, TensorFlow, Keras

ResNet50



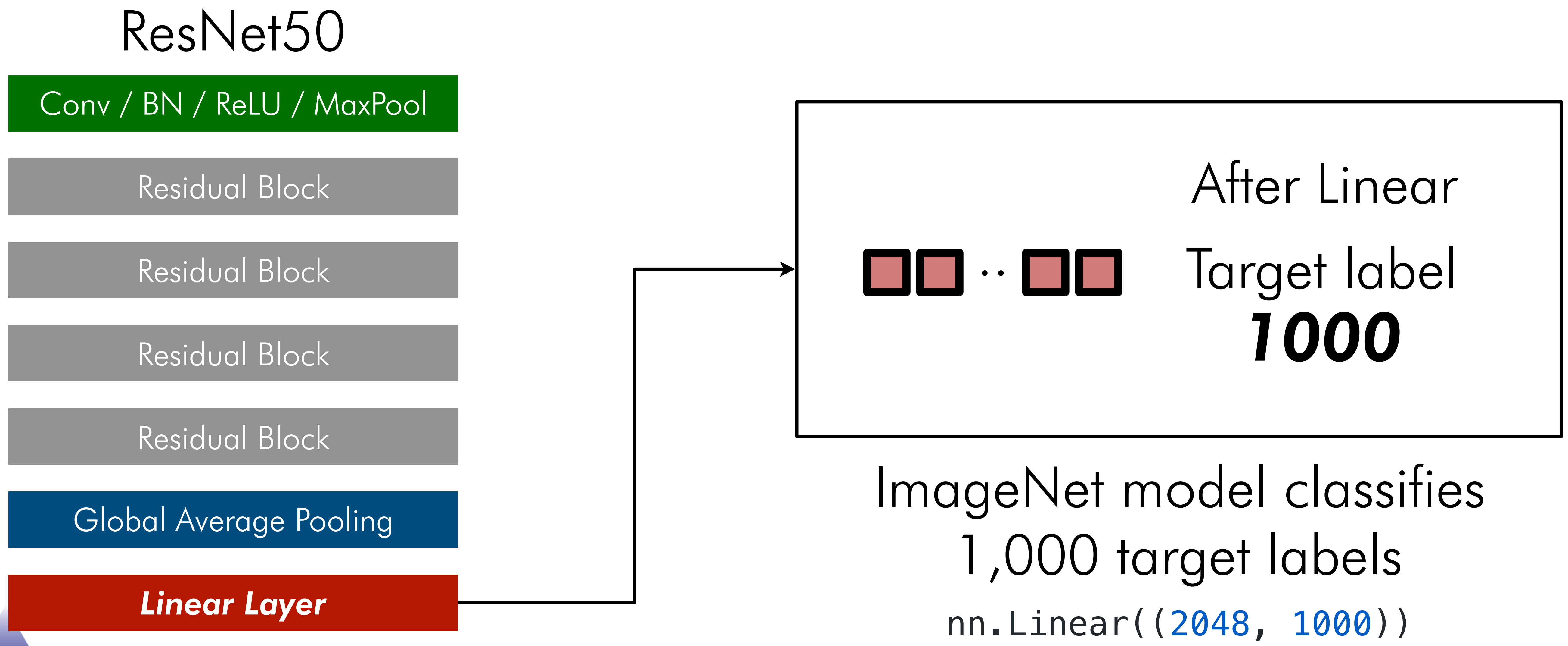


Global Average Pooling
shrinks height and width to 1

```
nn.AdaptiveAvgPool2d((1, 1))
```

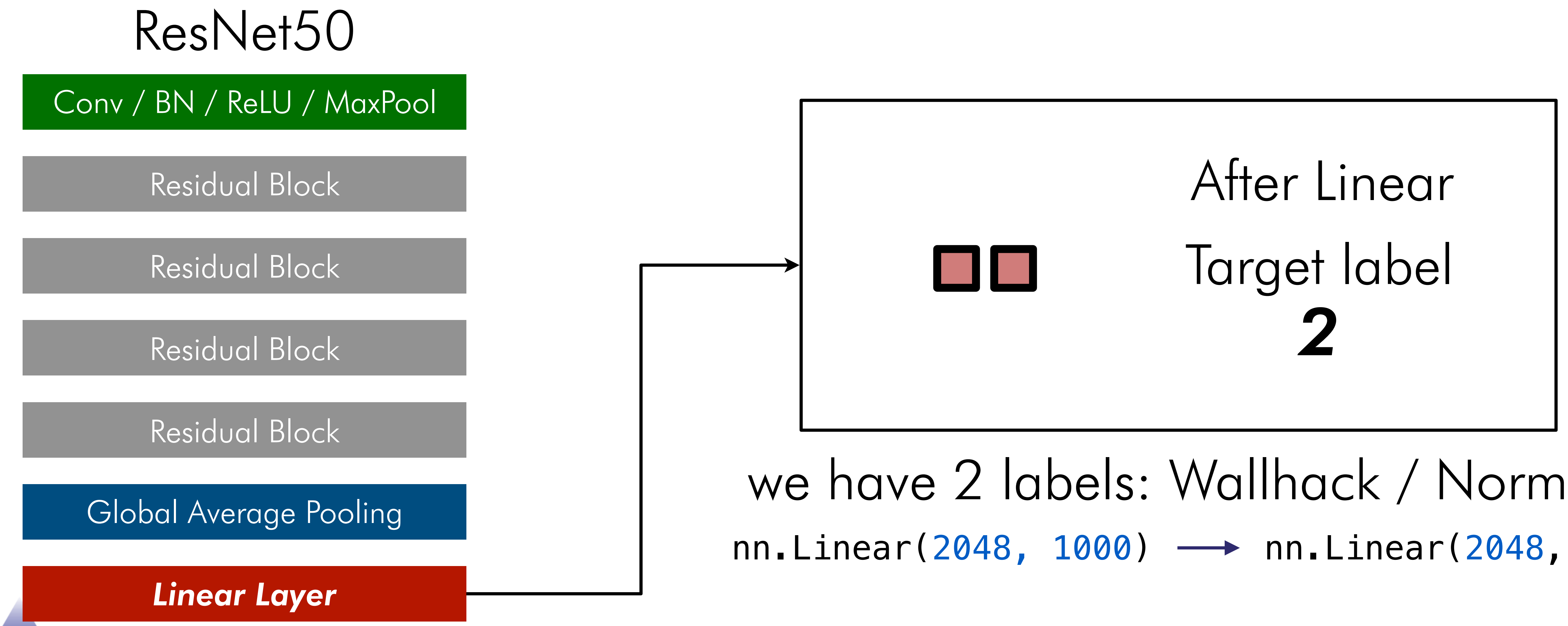
Pre-trained Layers + New Classifier

- Pre-trained models are available with PyTorch, TensorFlow, Keras



Pre-trained Layers + New Classifier

- Pre-trained models are available with PyTorch, TensorFlow, Keras



we have 2 labels: Wallhack / Normal

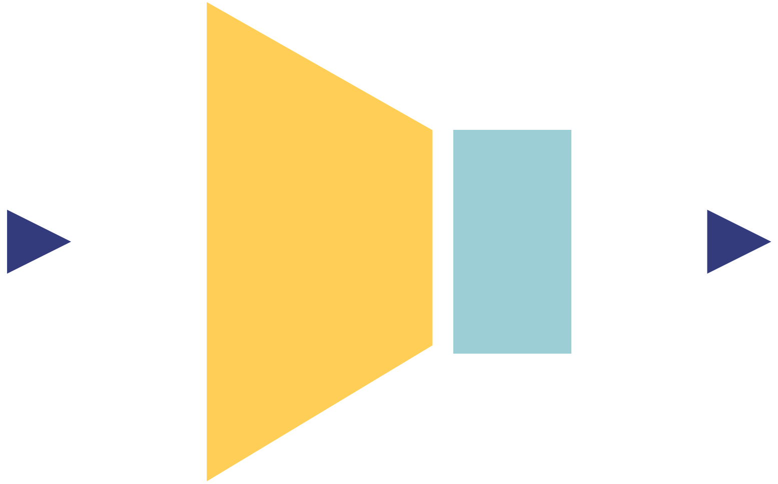
`nn.Linear(2048, 1000)` → `nn.Linear(2048, 2)`

Effect: Worked well!

- Pre-trained ResNet50: ***80%*** test accuracy with 2,000 images

Input

Fine-tuned ResNet50



Output

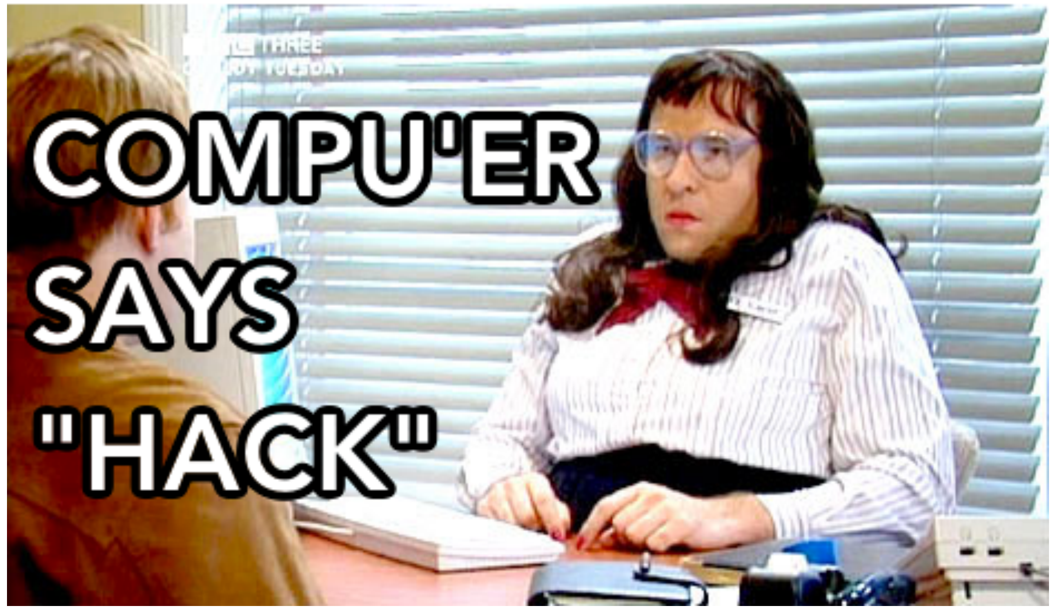
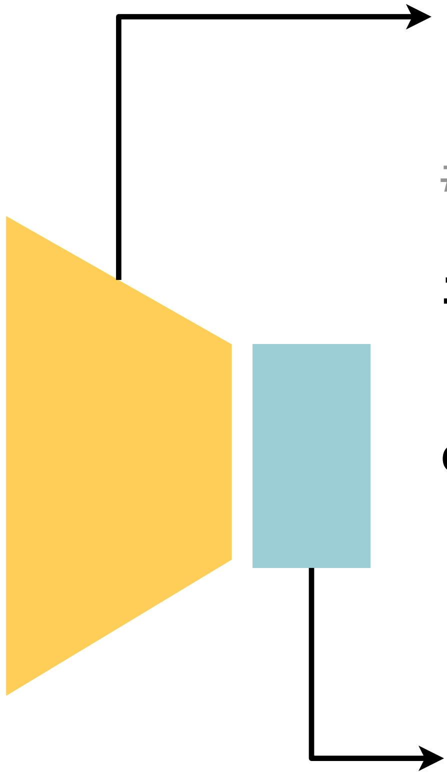


Image Credit: Little Britain

Effect: OK with 1 GPU

- Takes less than 20 minutes with a ***single*** NVIDIA 1060 GPU

Input



Freeze the parameters of the feature extractor

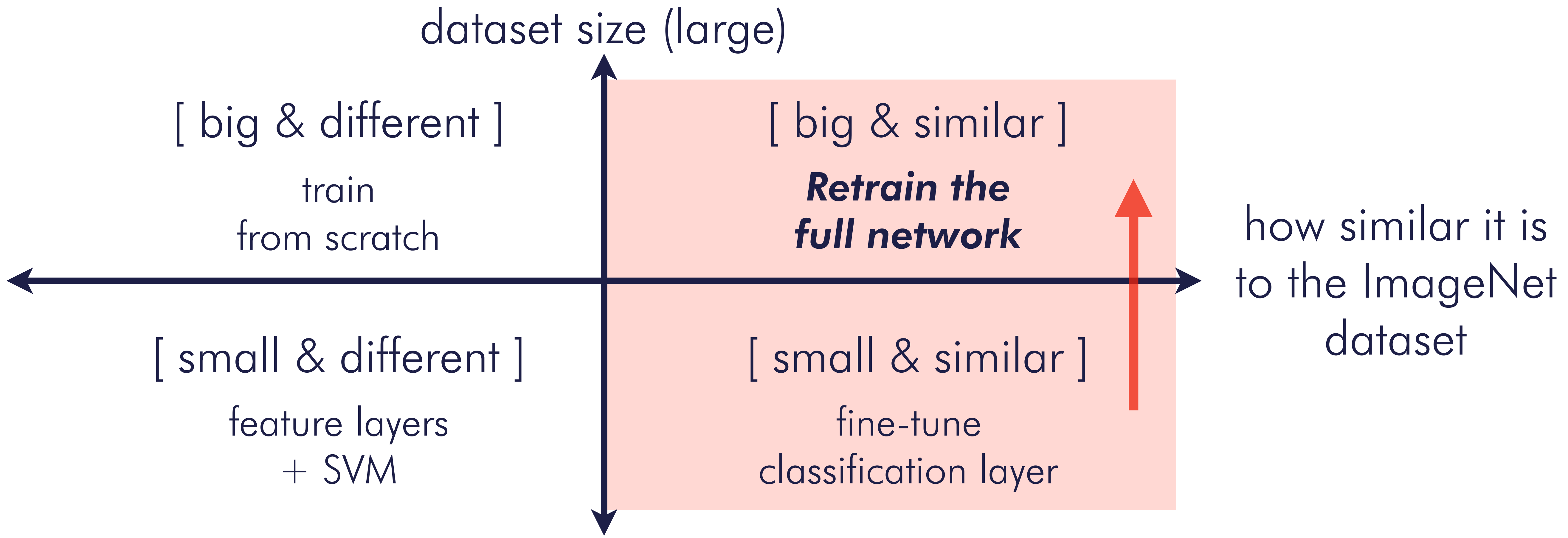
```
# Class CustomResNet

if toFreeze:
    for param in self.feature_extractor.parameters():
        param.requires_grad=False
else:
    for param in self.feature_extractor.parameters():
        param.requires_grad=True
```

Train the classifier only

Transfer Learning Strategies from CS231n

- ***Guideline*** based on the size and nature of your dataset
- We re-trained the full network after getting 10,000+ images



However: terribly overfitted

- Didn't work with ***unseen*** maps and weapons
- Too many image features to learn

What we wanted

```
function(seeing thru wall) = "HACK"
```

What we actually got

```
function(cool golden weapon) = "HACK"
```

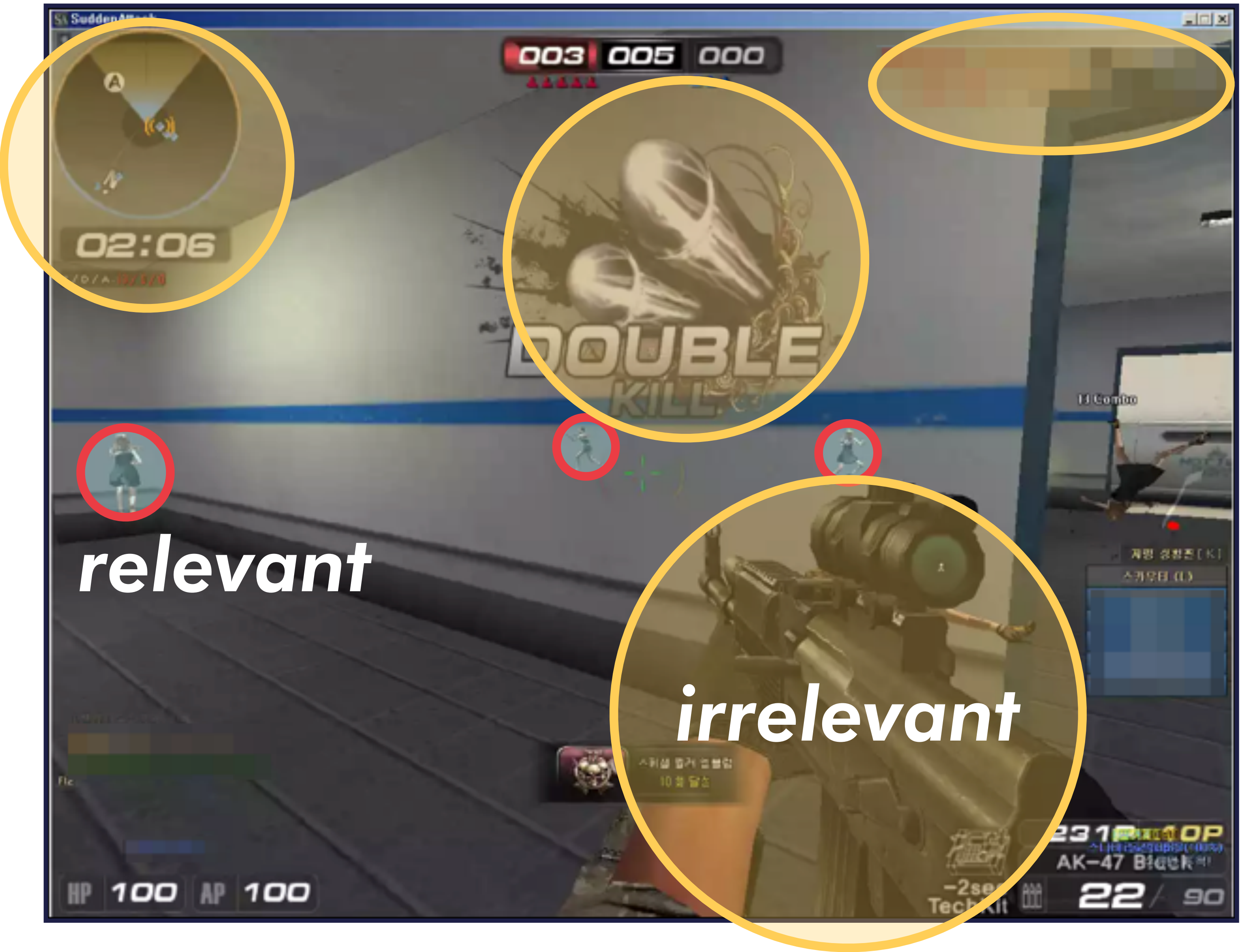
vs. Limited Signal

Problem:

003
005
000
02:06
DOUBLE
KILL
relevant
HP 100
AP 100
AK-47
-2sec
TechKit
22
90

Problem: Low Signal / Noise Ratio

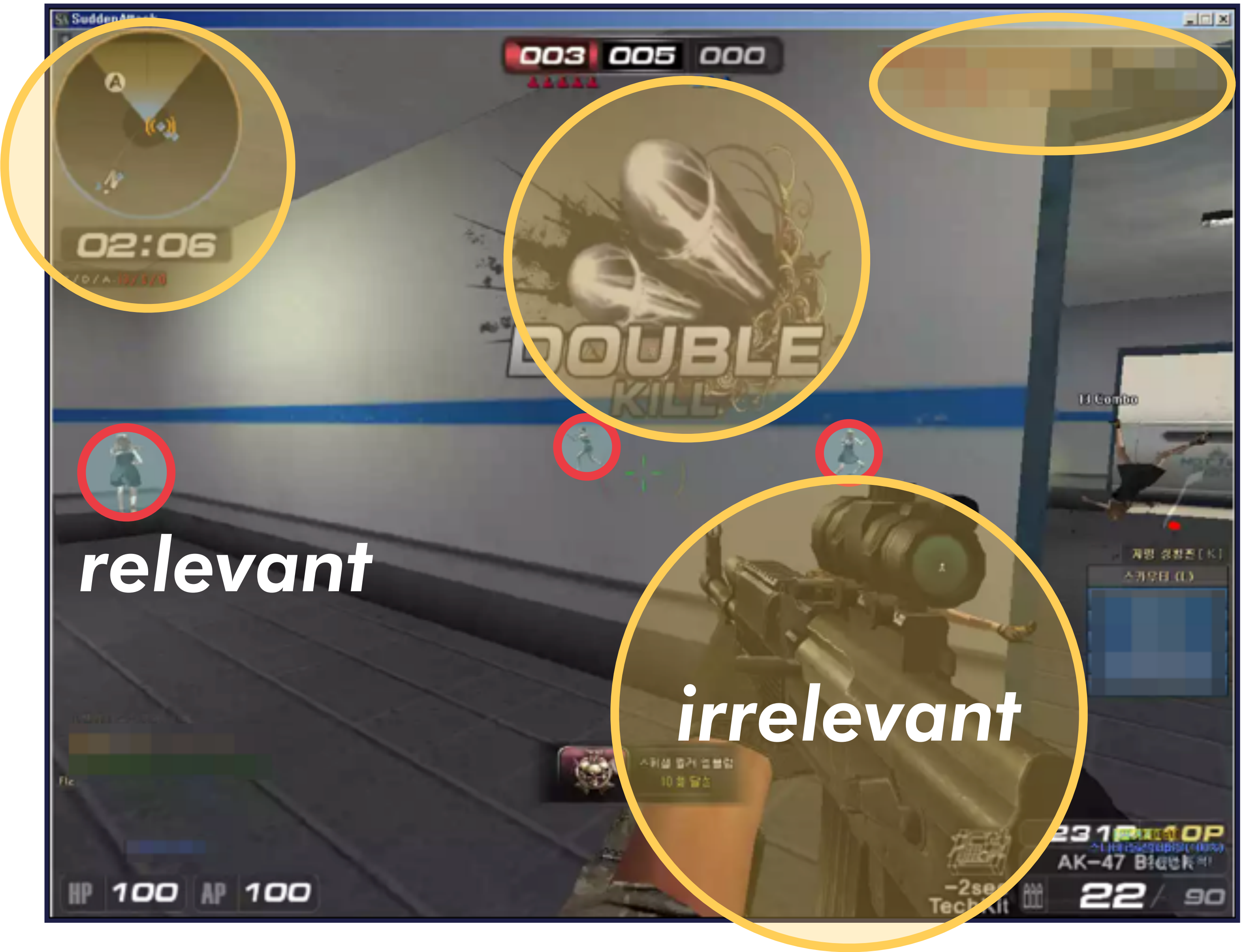
- Too many irrelevant features spoil the training
- Model predicts based on kill marks or weapons ***NOT*** on wallhack figures



Compared to other classification tasks

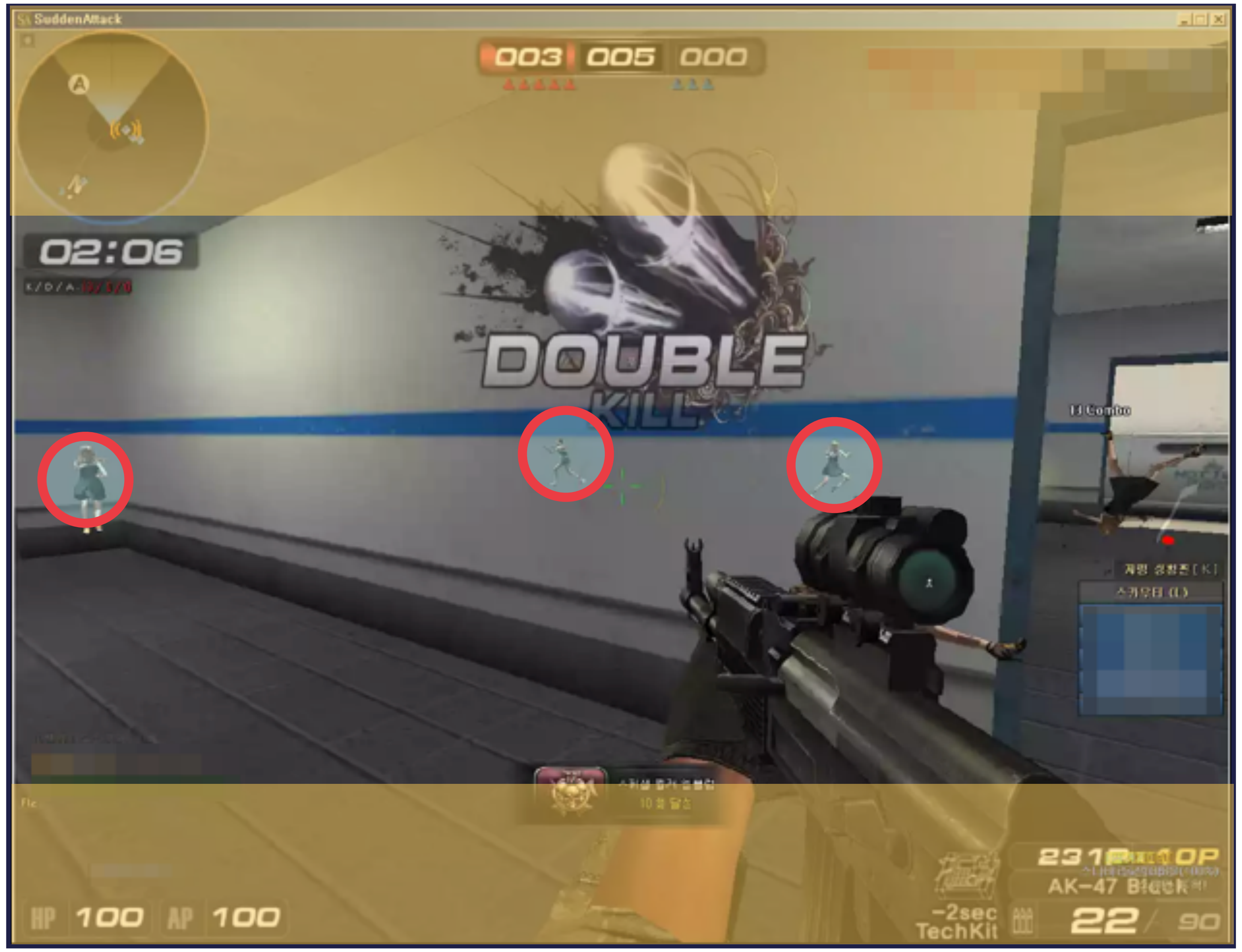
MNIST

ImageNet



Solution: Divide & Conquer in Patches (1)

- 1) ***Remove Top & Bottom***
  - SA maps are mostly flat
  - Players tend to place targets on the line of the crosshair



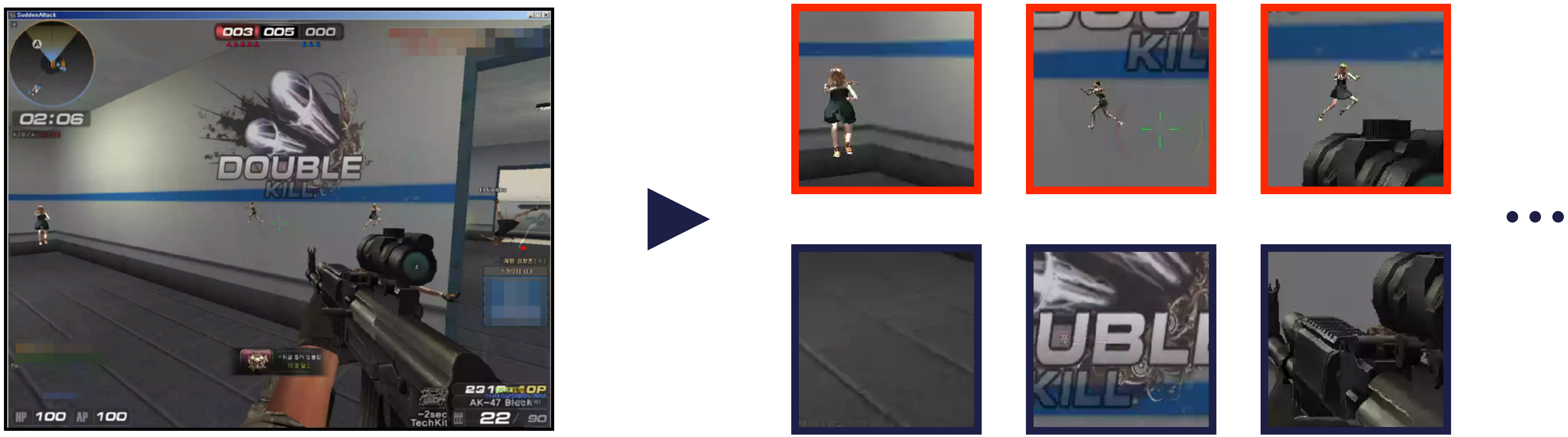
Solution: Divide & Conquer in Patches (2)

- 1) Remove Top & Bottom
- 2) ***Break into patches***
  - found optimal # of patches via experiments
  - single 960x540 image > multiple 197x197 patches

Effect: Less prone to Noise

Effect: the more data the merrier

- Generate 24+ patches from a single image



Downside: labelling all over again

- Re-labelled 5,000 wallhack / normal patches

...

Image Credit: The Simpsons

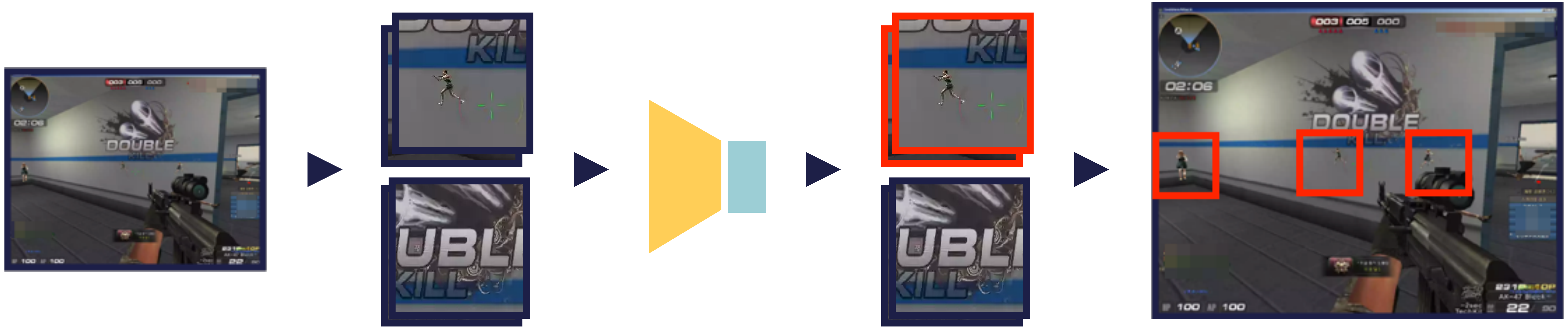
Effect: worked superbly with ResNet50

- Test accuracy: **92%**

patches *ResNet50*

Effect: worked superbly with ResNet50

- Test accuracy: 92% + kind of ***object localisation effect***

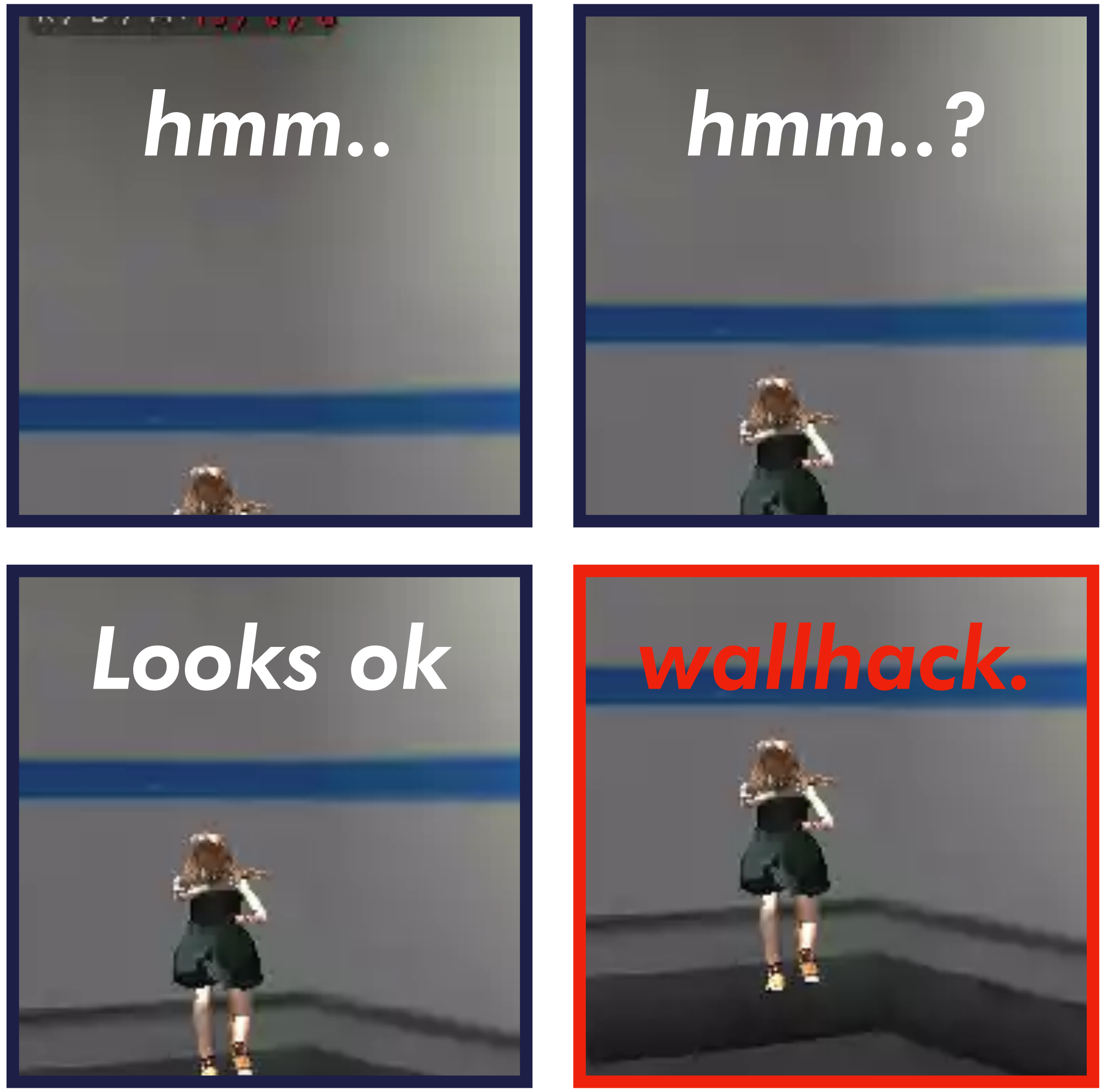


patches *ResNet50* *Draw boxes on hack patches*

Example

Downside: Confusing patches

- Figures on edges
  side effect of discretisation
- Even ***human*** inspectors
  find it difficult to tell



Same image region
with different crop coordinates

Downside: *still makes mistakes*

- Patch-wise inference helps but does not tell ***why***

vs. Limited Trust

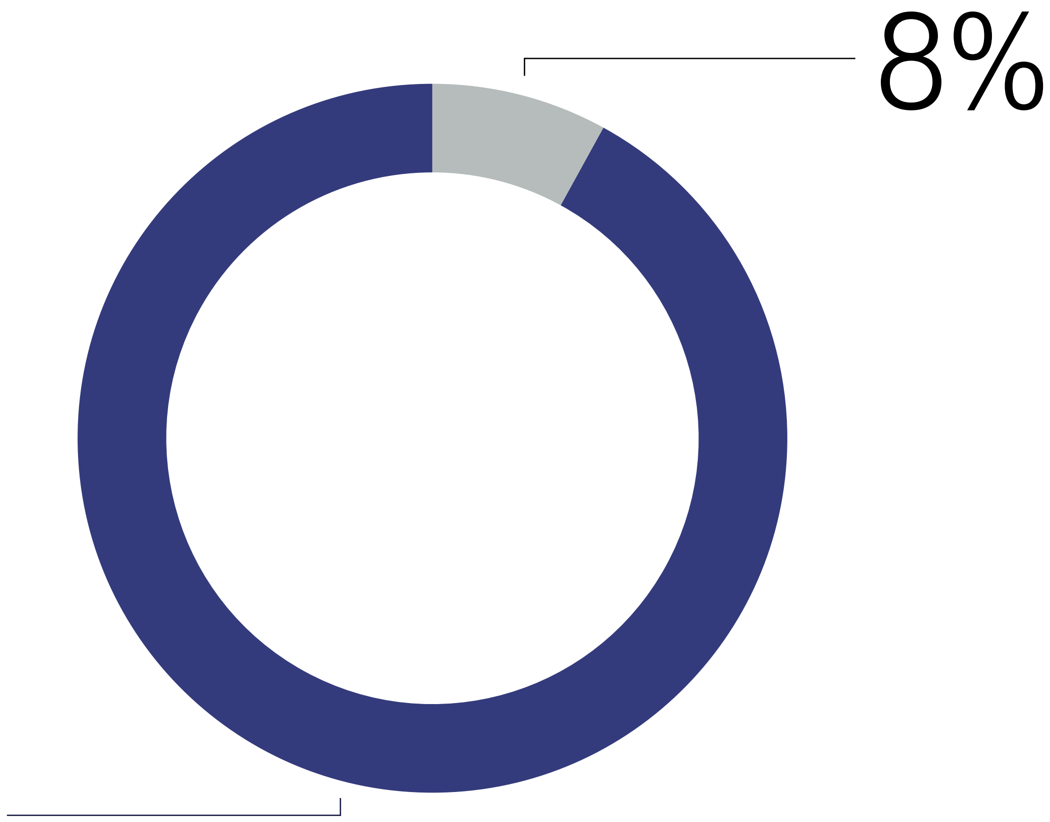
Hey everyone, it WORKS!!!

- Test accuracy 92% is amazing enough***!***

ML team:

Look, we made it!

92%

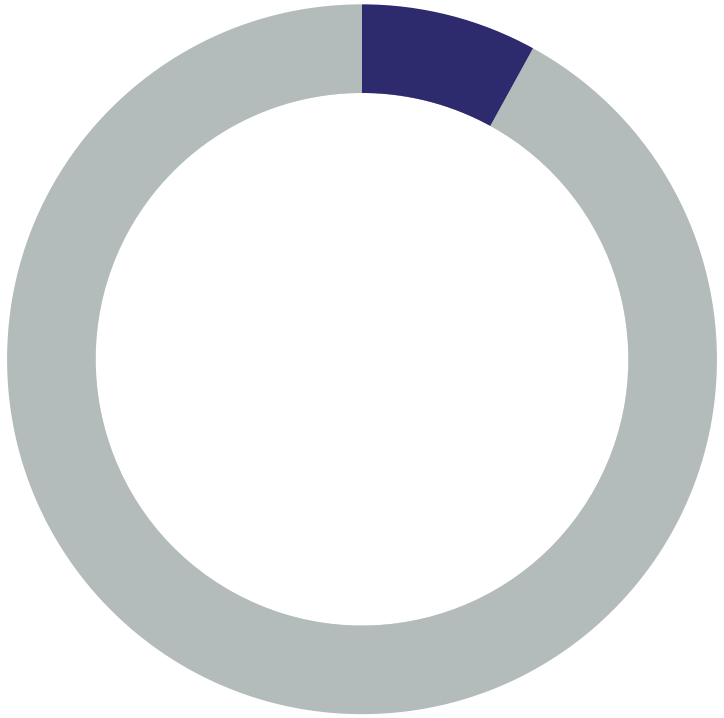


Hey everyone, it WORKS?

- Test accuracy 92% is amazing enough**?**

8%

Community manager:
Even **99%** is
not good enough.



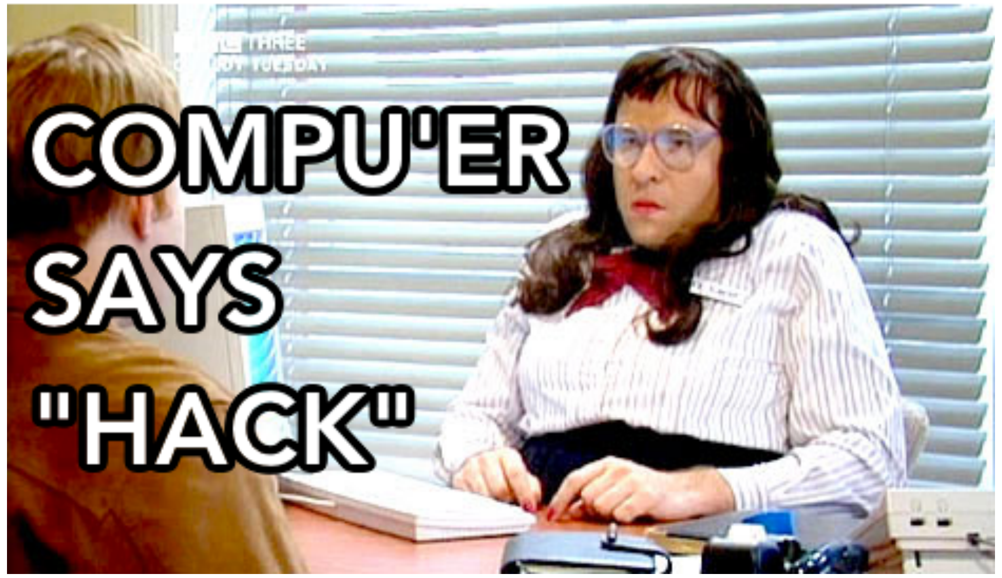
ML team:
Look, we made it!

92%

Same goal, Different approaches

- ML team explores new algorithms
- If it works, being a blackbox model is not a big deal

Model



ML guy

Model

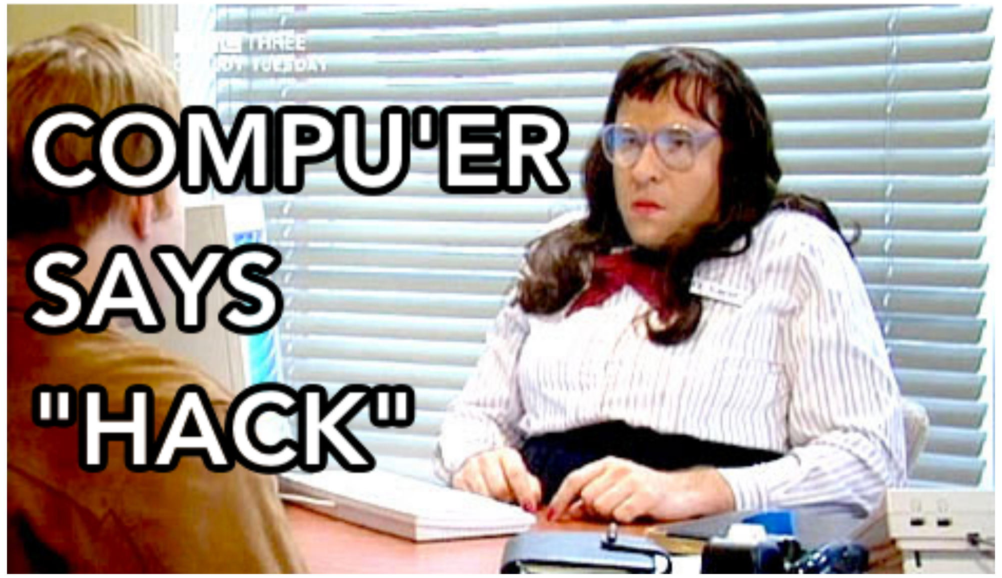
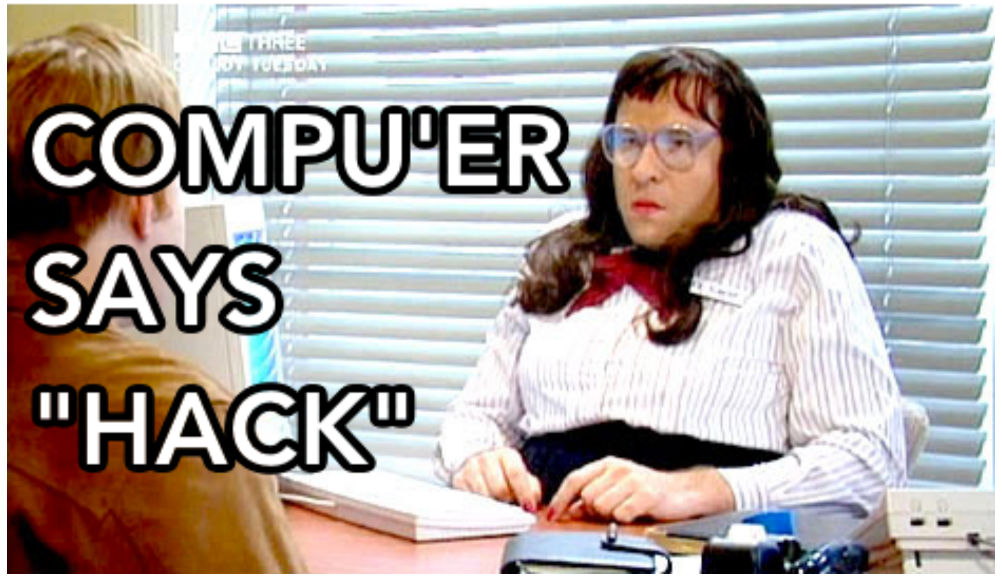


Image Credit: Little Britain

Same goal, Different approaches

- Community managers face gamers directly
- ***Distrust*** comes from not knowing why

Community manager



Gamer

I got banned for no reason	1 post	Jan 9, 2019
Got banned for no reason.	1 post	Dec 19, 2018
Literally banned for no reason...	1 post	Sep 15, 2018
Banned For No Reason	6 posts	Aug 8, 2018

More results from www. .com

Community manager

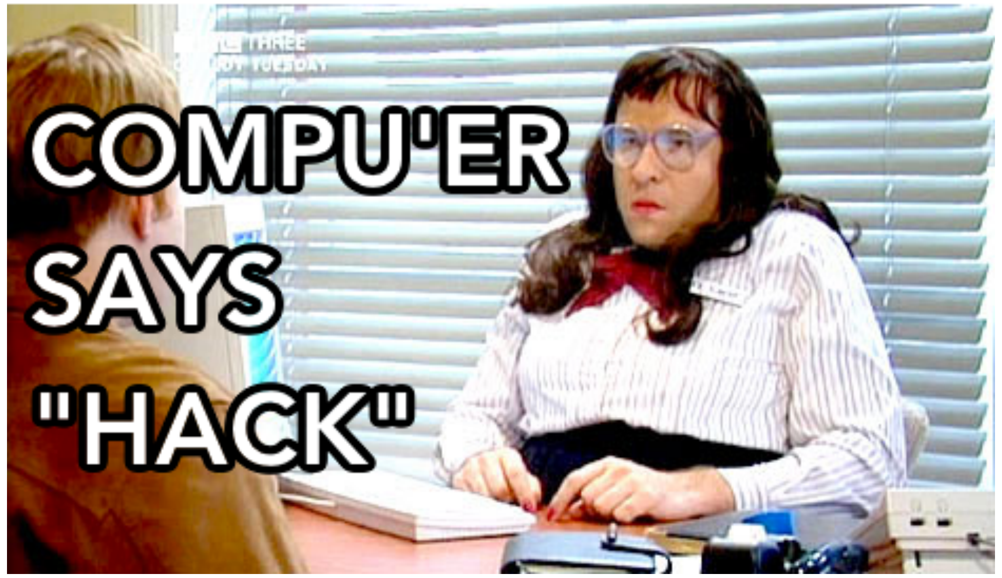
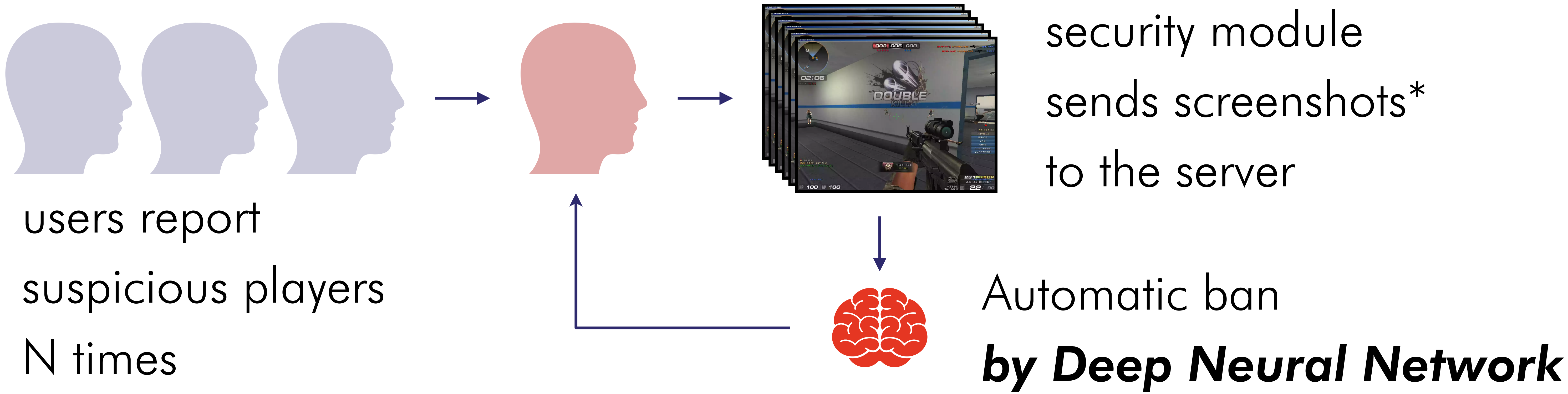


Image Credit: Little Britain

We're not there to show off

- But to make things easier and ***solve problems***



DL to enhance human productivity

- Filter out normal images as much as possible

users report
suspicious players
N times

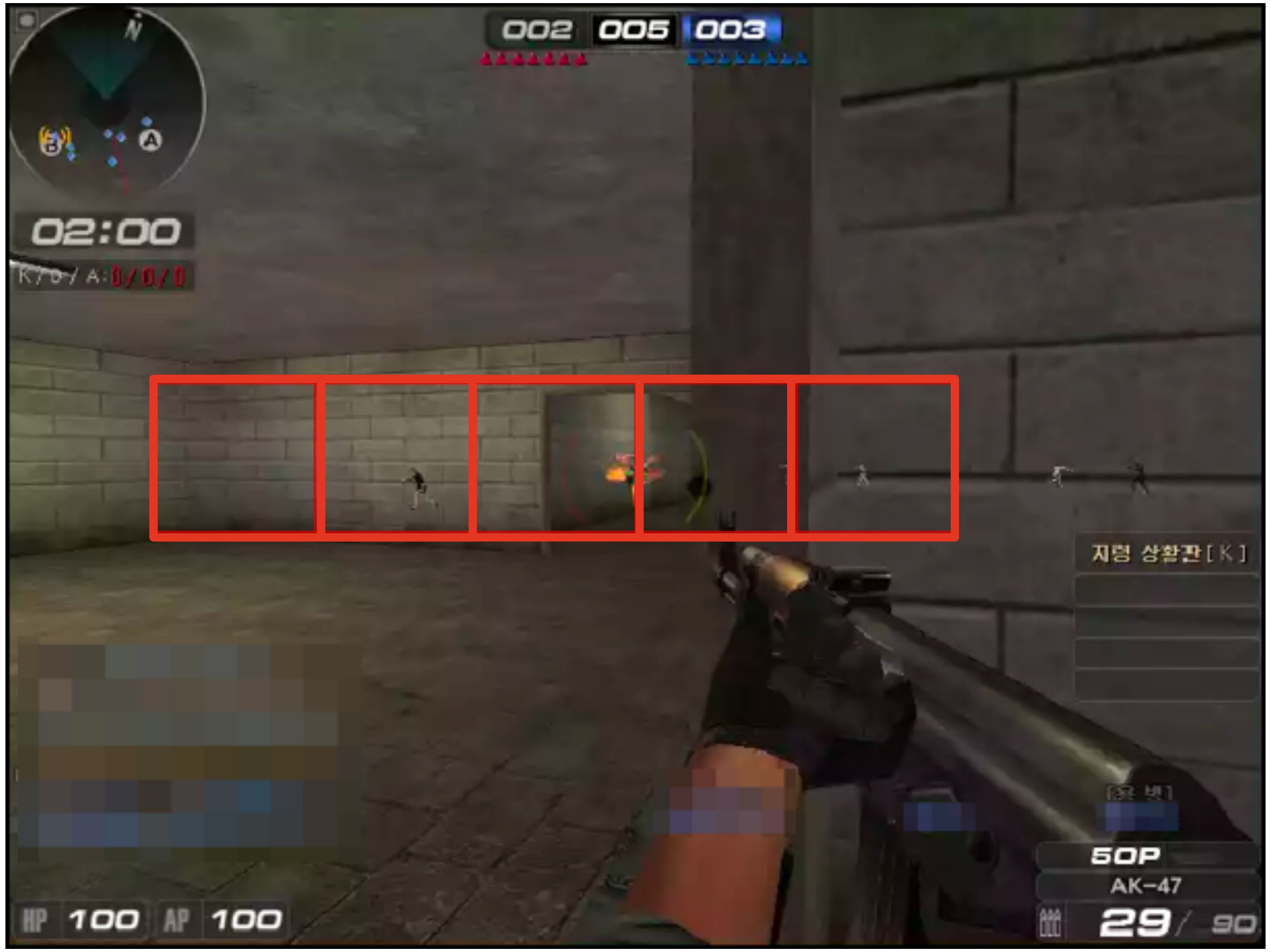
security module
sends screenshots*
to the server

Automatic classification
by Deep Neural Network

ban abusers
via final manual inspection

Problem: More Accurate Bounding Boxes

- Bounding boxes shorten per-image inspection time
- Can wallhack localisation be ***more accurate***?



Solution: Class Activation Map (CAM)

- **CAM** tells where the model look at for its prediction

"Learning Deep Features for Discriminative Localization"
by Zhou et al (2016)

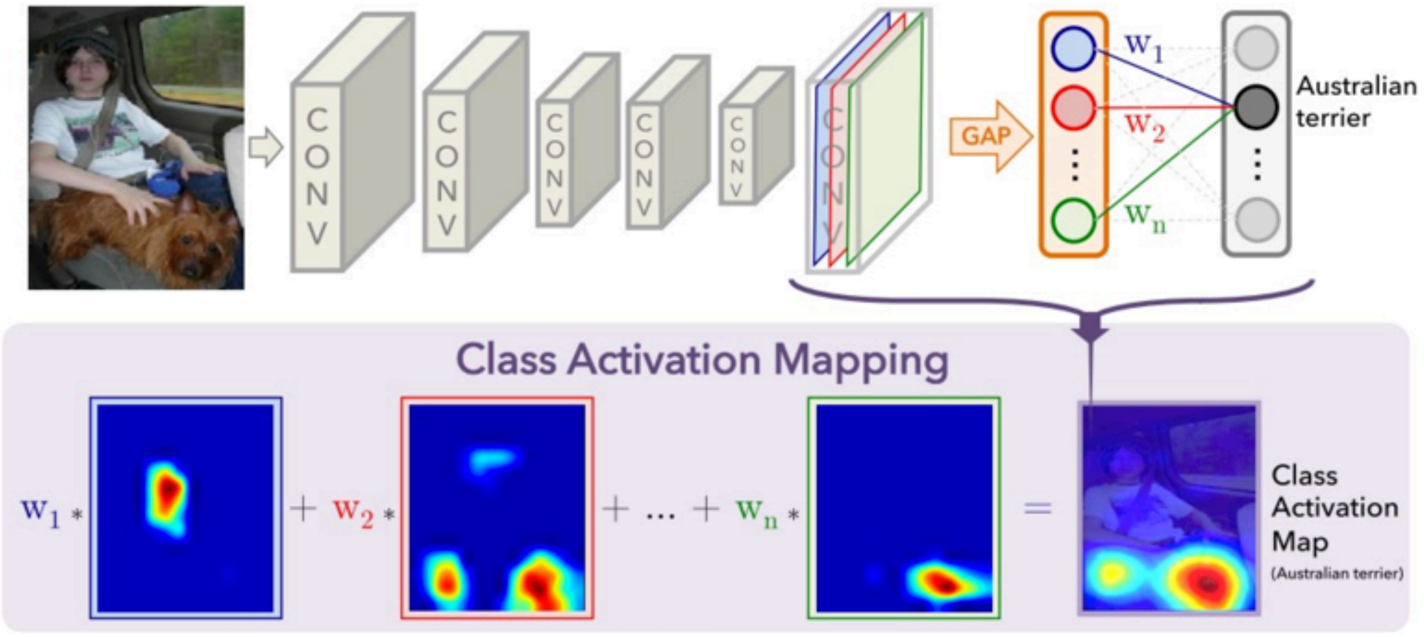
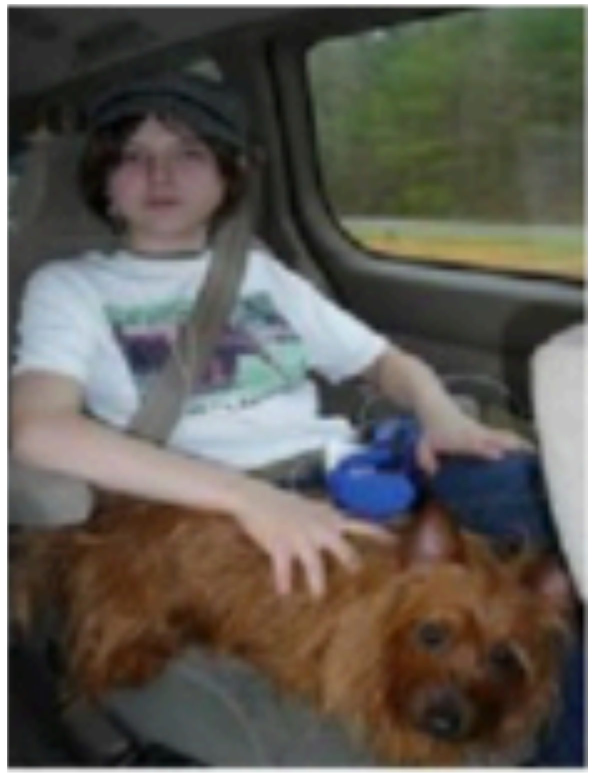


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.



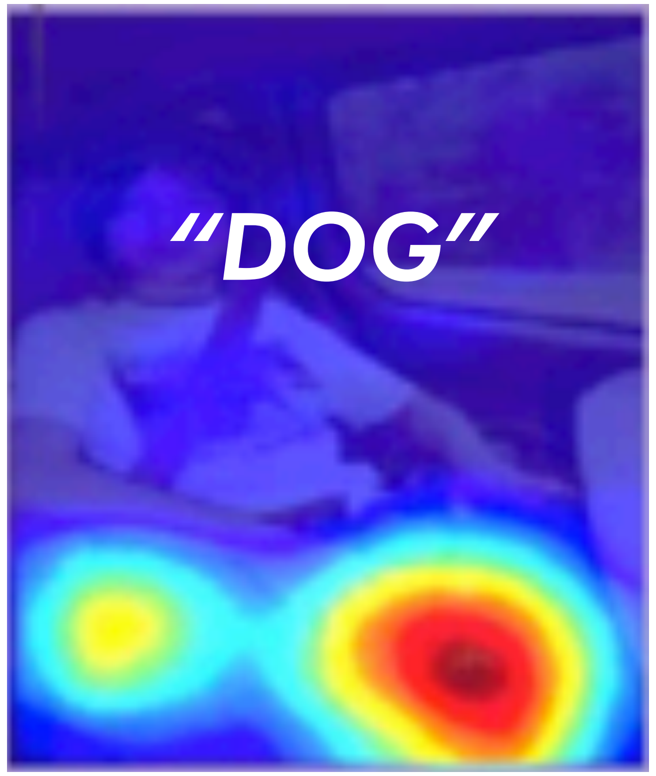


Image Detection without bounding boxes

- ***Weakly*** Supervised Learning: works only with image-level labels
- Bounding box coordinates are expensive to prepare

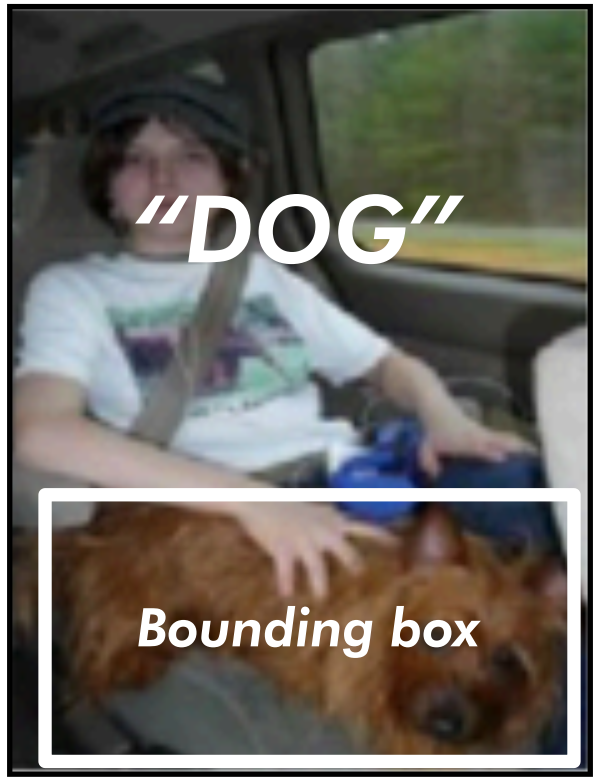
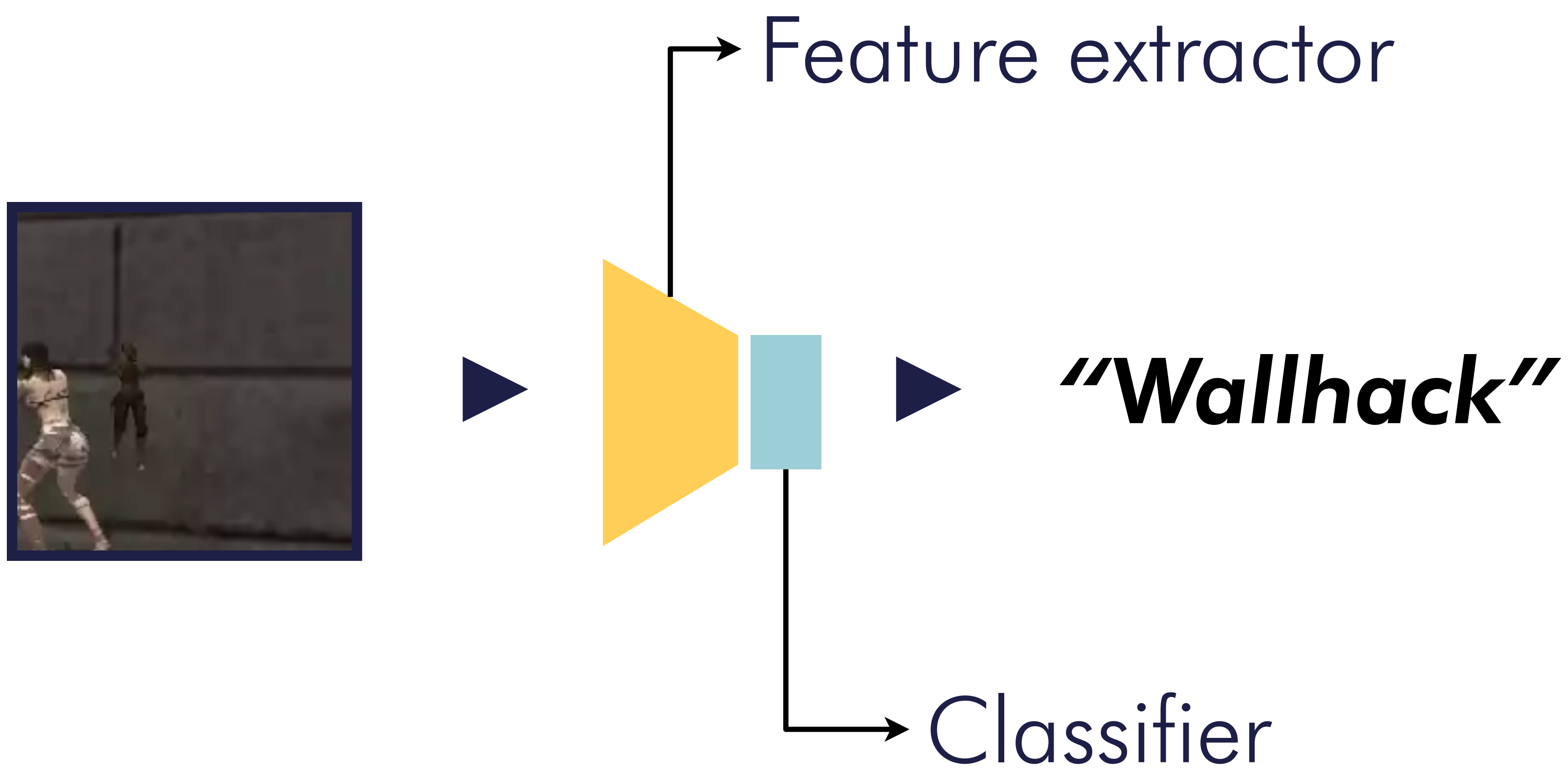


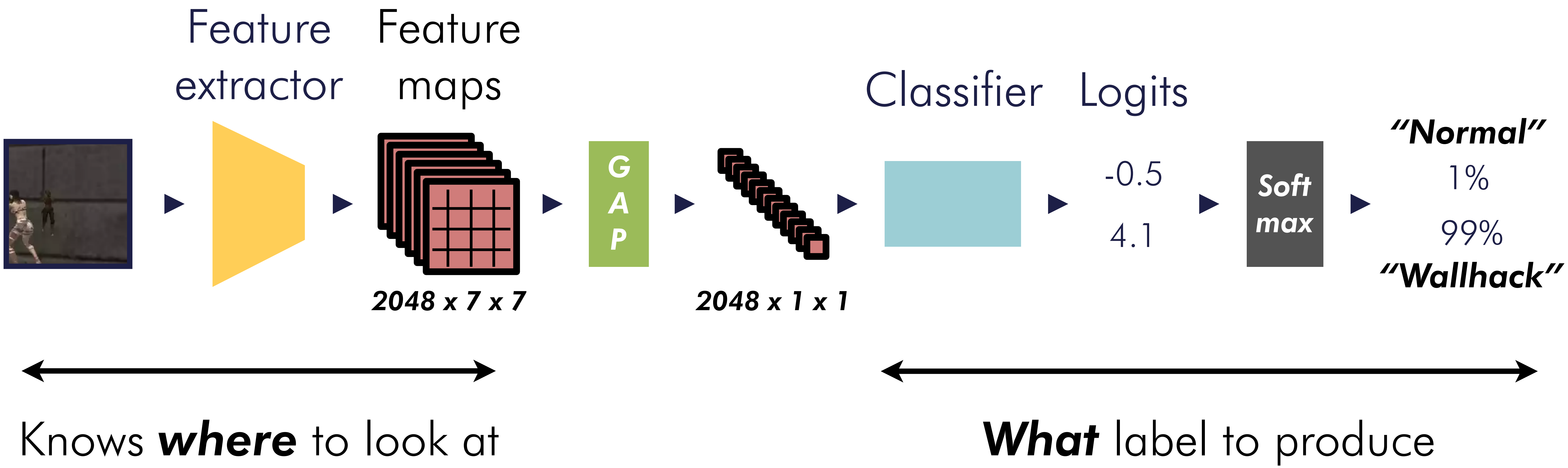
Image-level Label
Bounding Box coordinates (x, y, width, height)

Image-level label

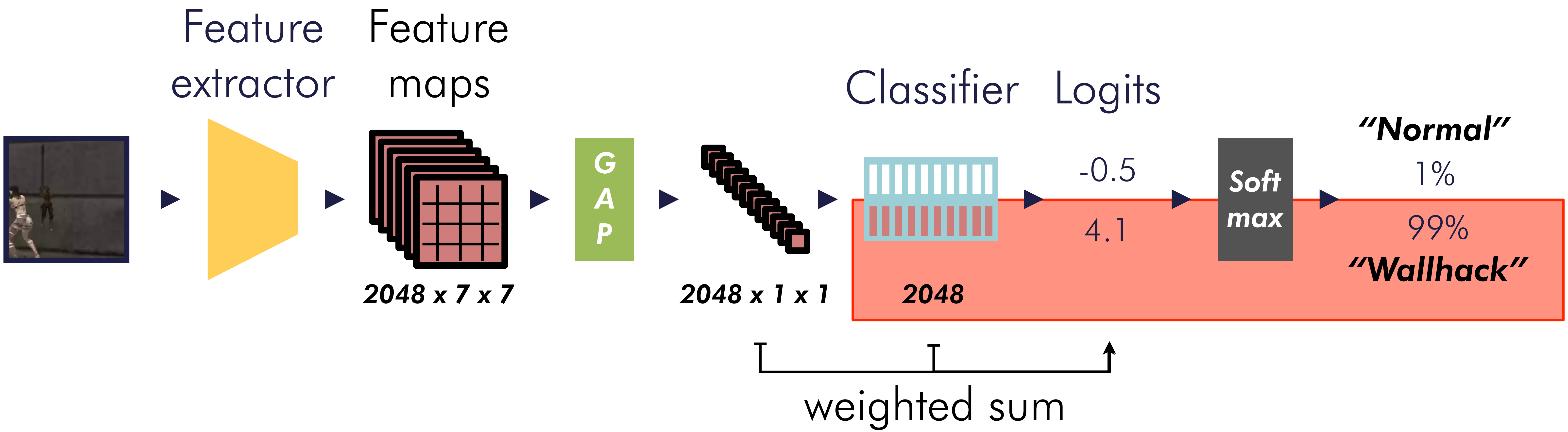
How CAM works



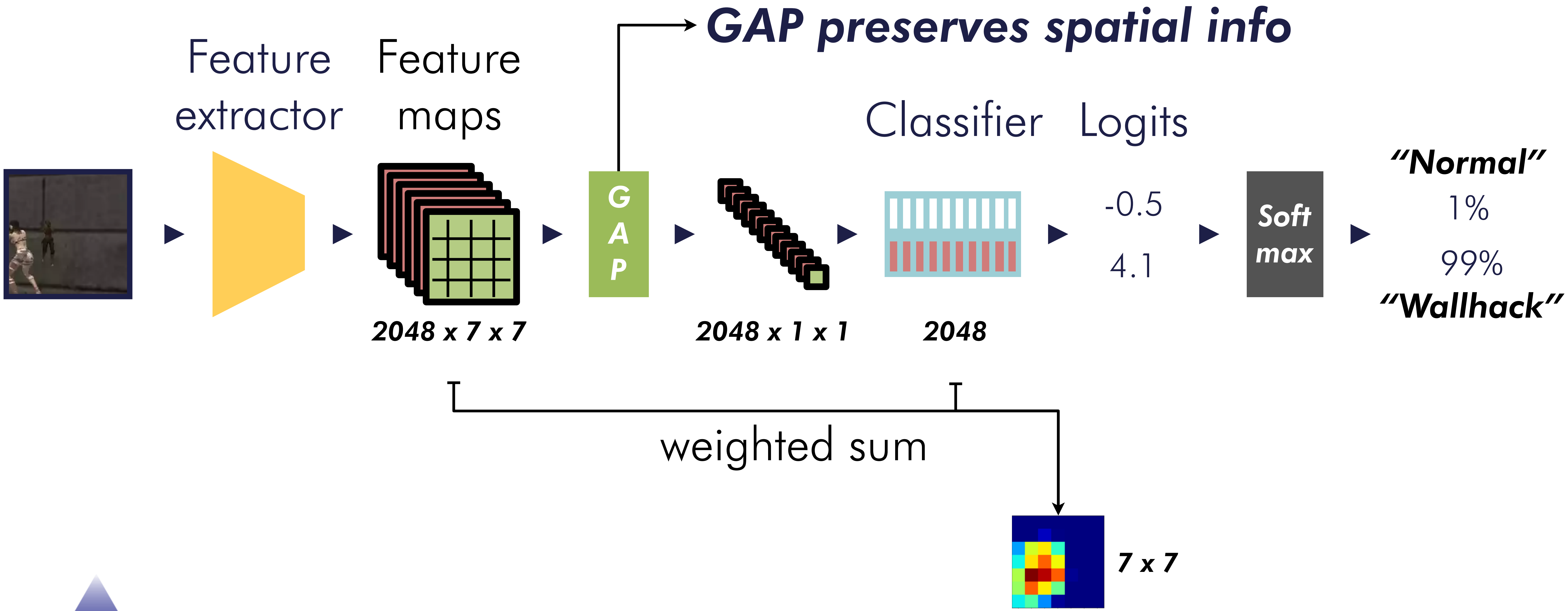
How CAM works



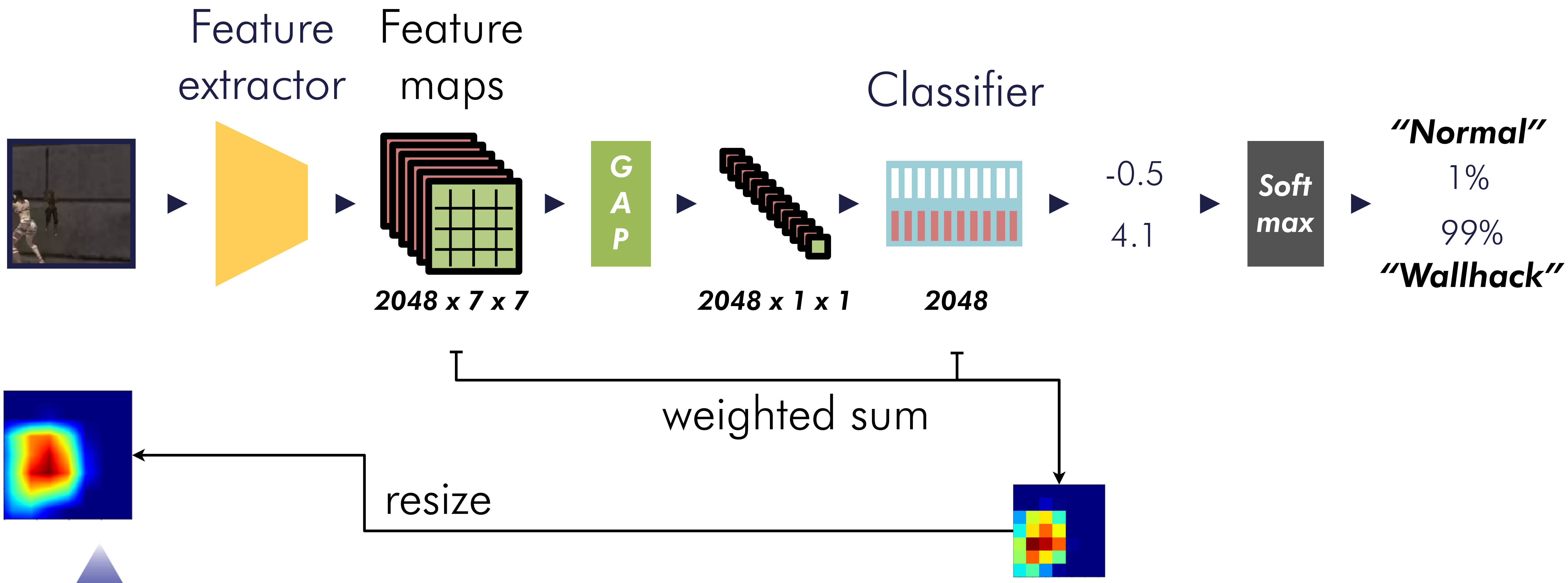
How CAM works



How CAM works



How CAM works

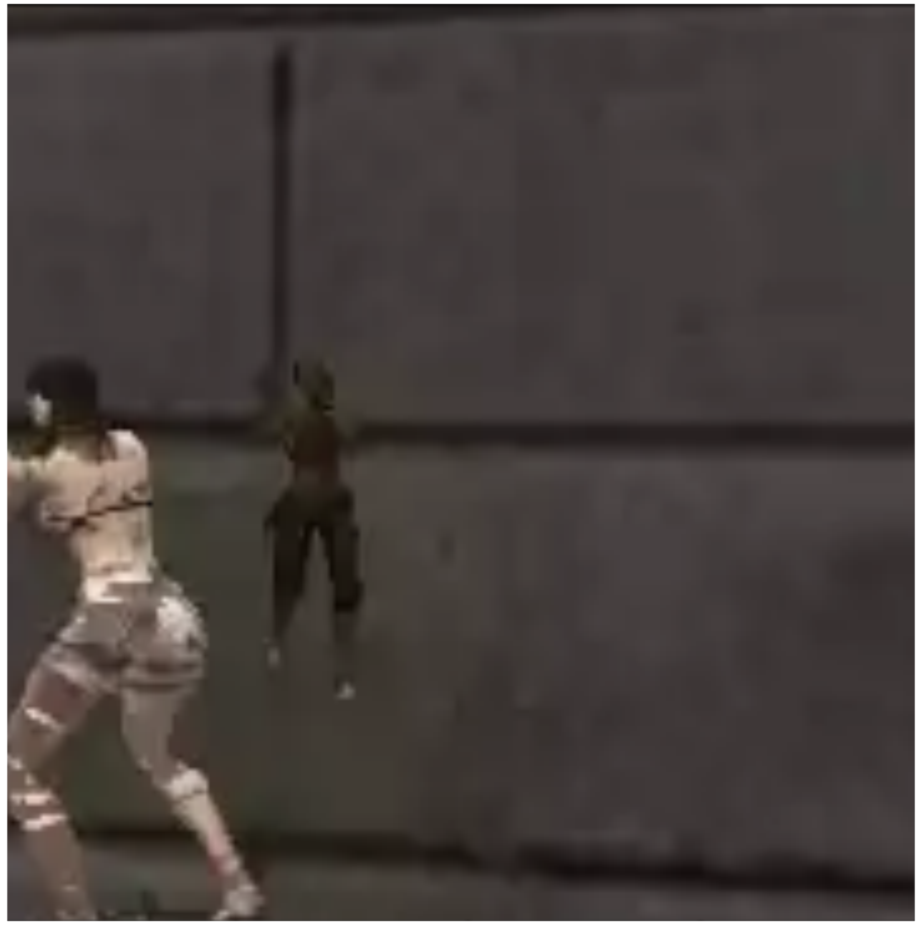


Drawing Bounding Box with CAM

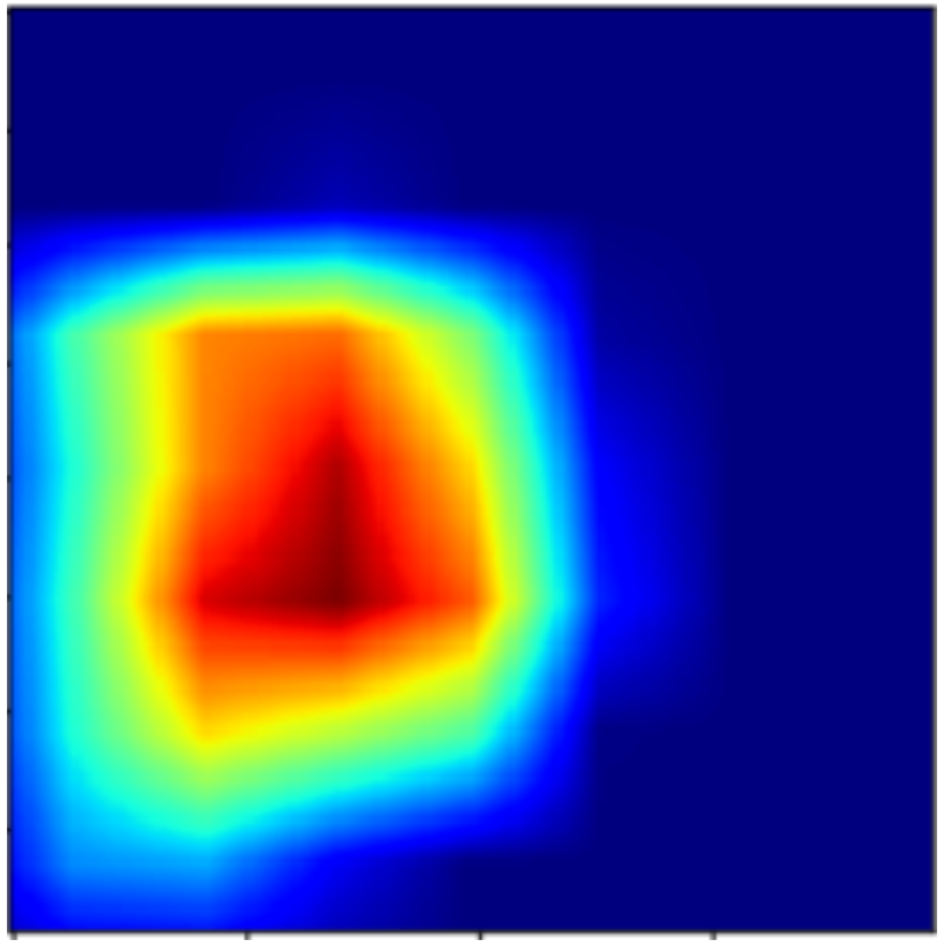
input patch

Drawing Bounding Box with CAM

input patch

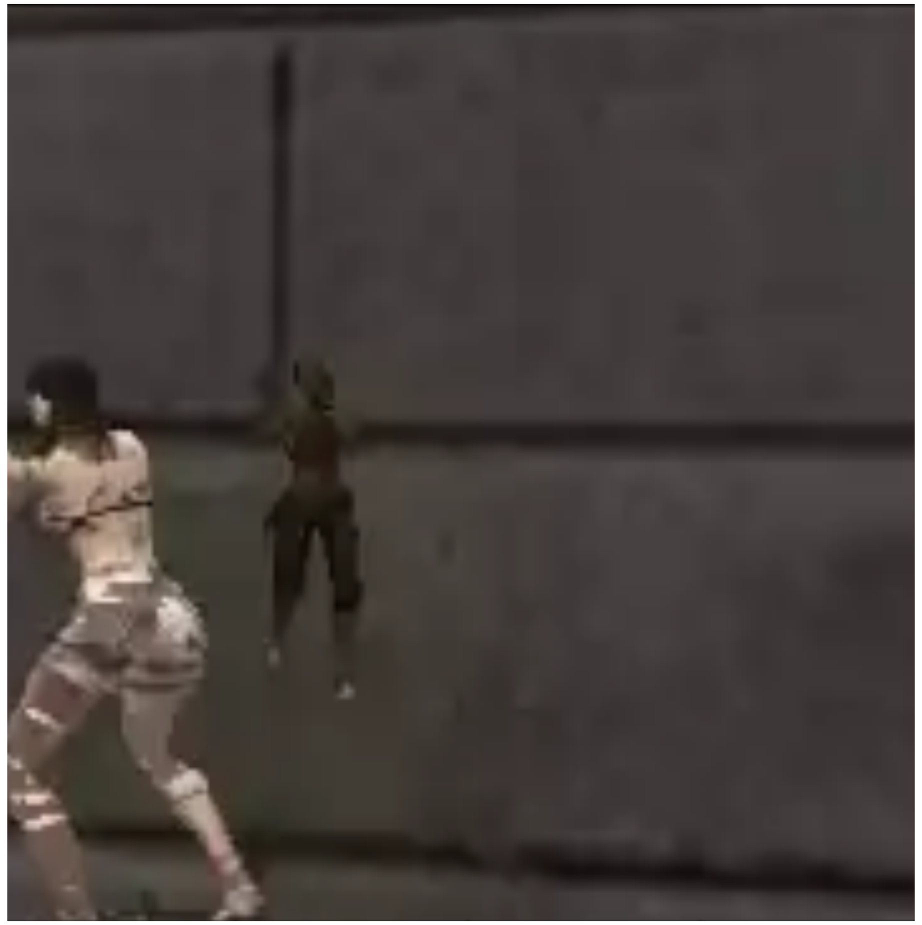


CAM heat map

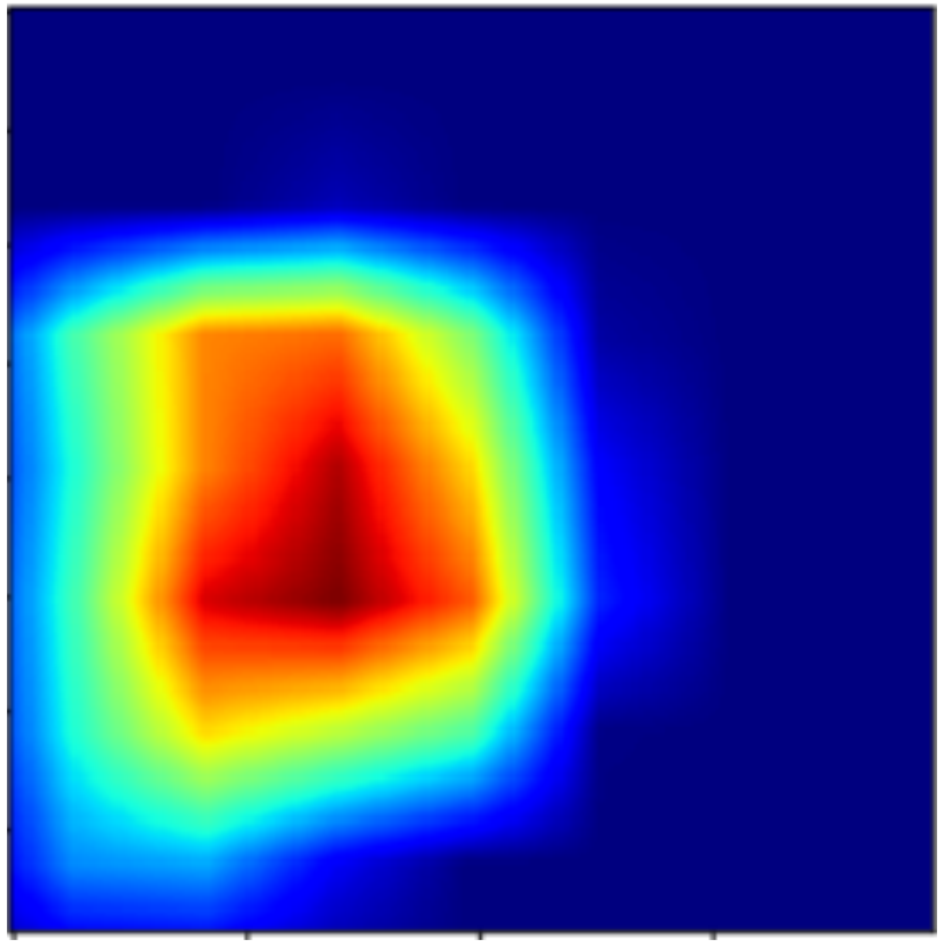


Drawing Bounding Box with CAM

input patch

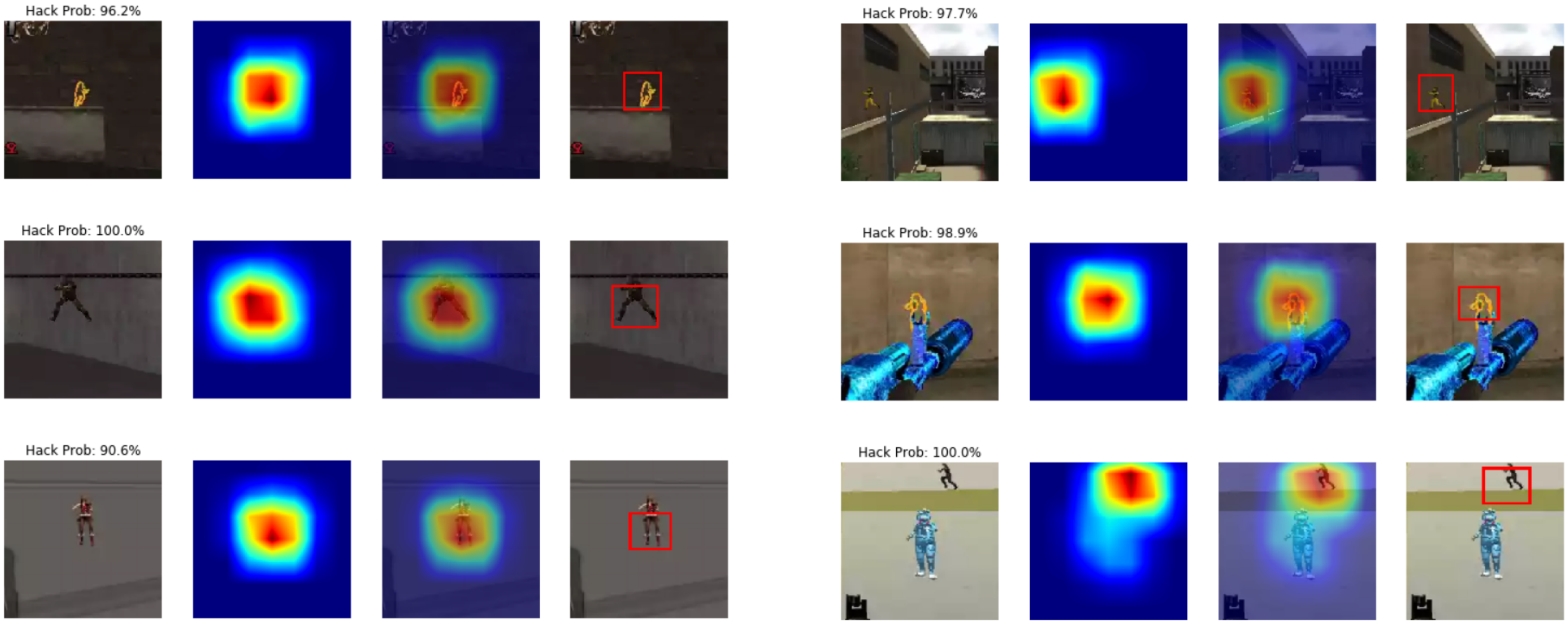


CAM heat map



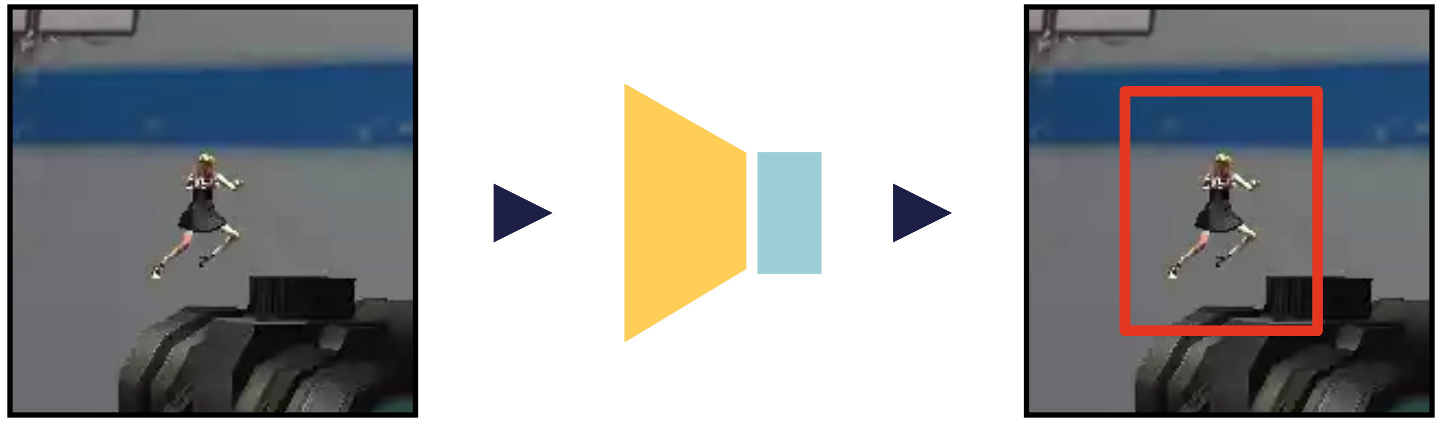
CAM bounding box

Effect: CAM result



Patch-wise Operation

- Input: patch
- Output: predicted label & bounding box

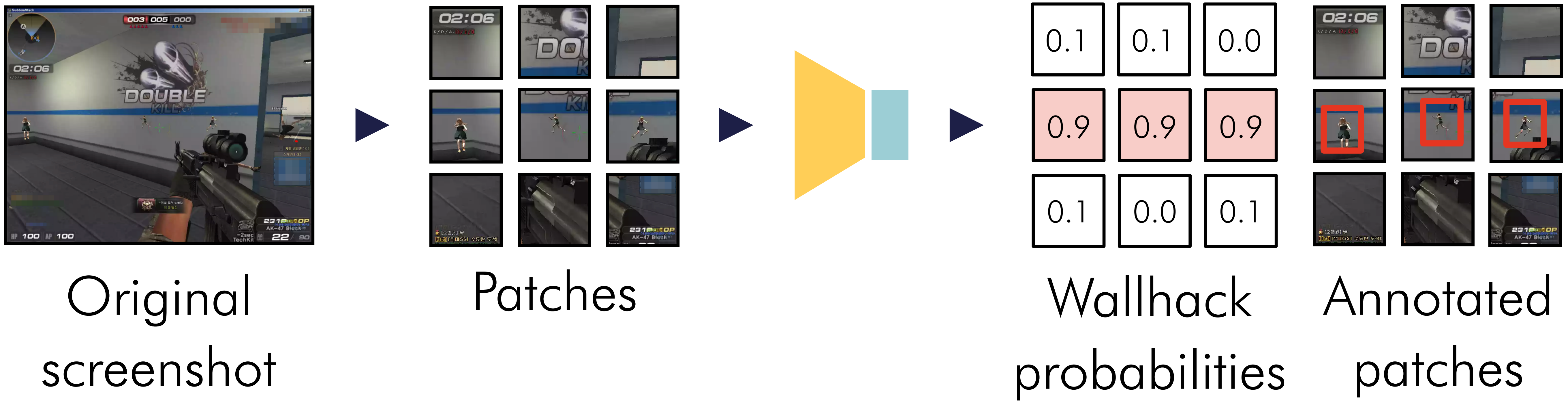


Patch "Wallhack"

More Efficient Inference

Patch-wise Operation for Screenshot

- Crop the original screenshot into patches for the model
- Get a ***Map*** of probabilities and ***CAMs***



Patch-wise Operation for Screenshot

- Enables more ***conservative*** classification
- Stitching CAMs together to annotate the screenshot

0.1	0.1	0.0
0.9	0.9	0.9
0.1	0.0	0.1

Wallhack probabilities

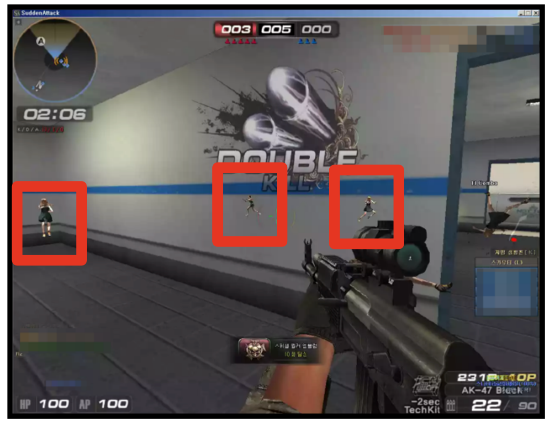
▶ ***Wallhack***

```
# hyper parameters for
# conservative decisions

wallhack_threshold = 0.5
number_of_hack_patches = 2
```

Annotated patches

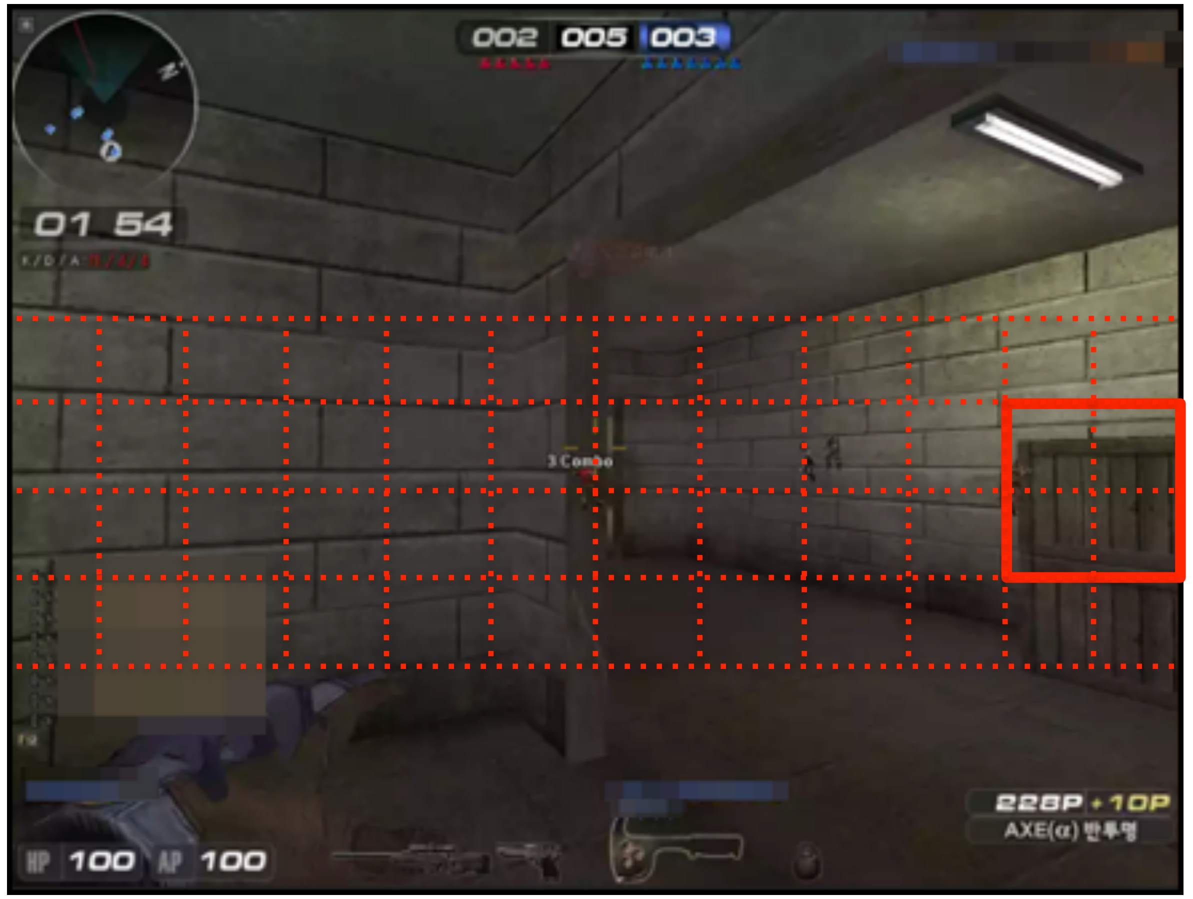
▶

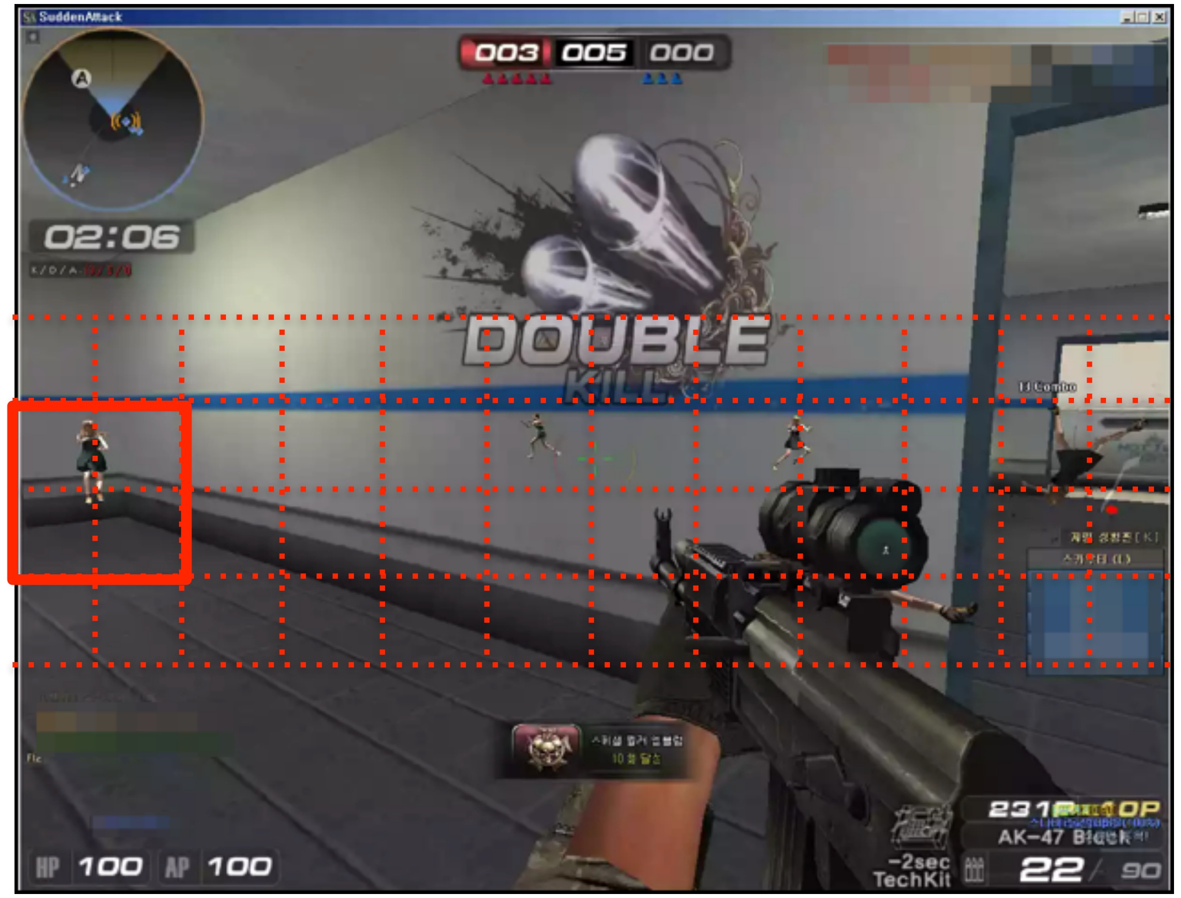


Annotated screenshot

Problem: Unclear Patches

- Patch cropping might ***miss*** the wallhack figures





Problem: Unclear Patches

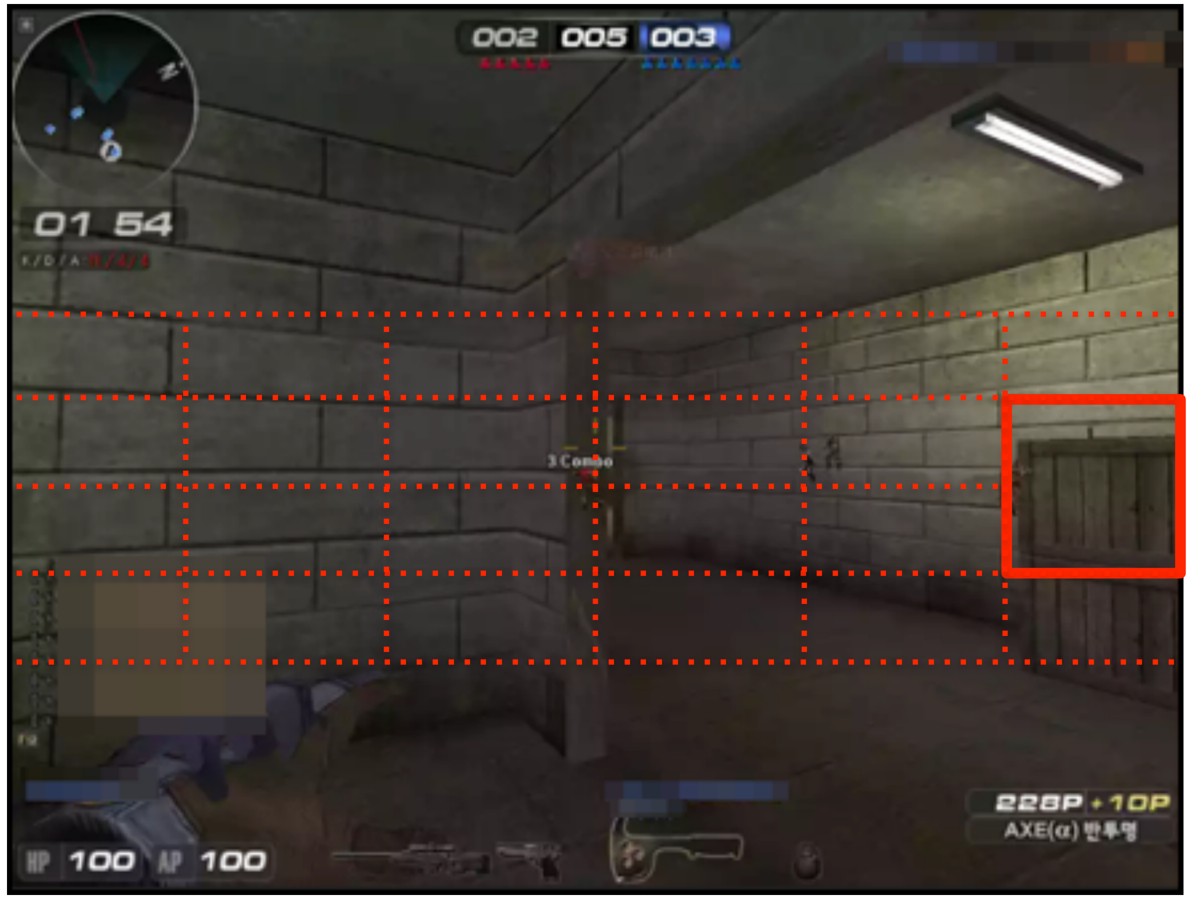
- ***Ambiguous*** wallhack figures due to cropping

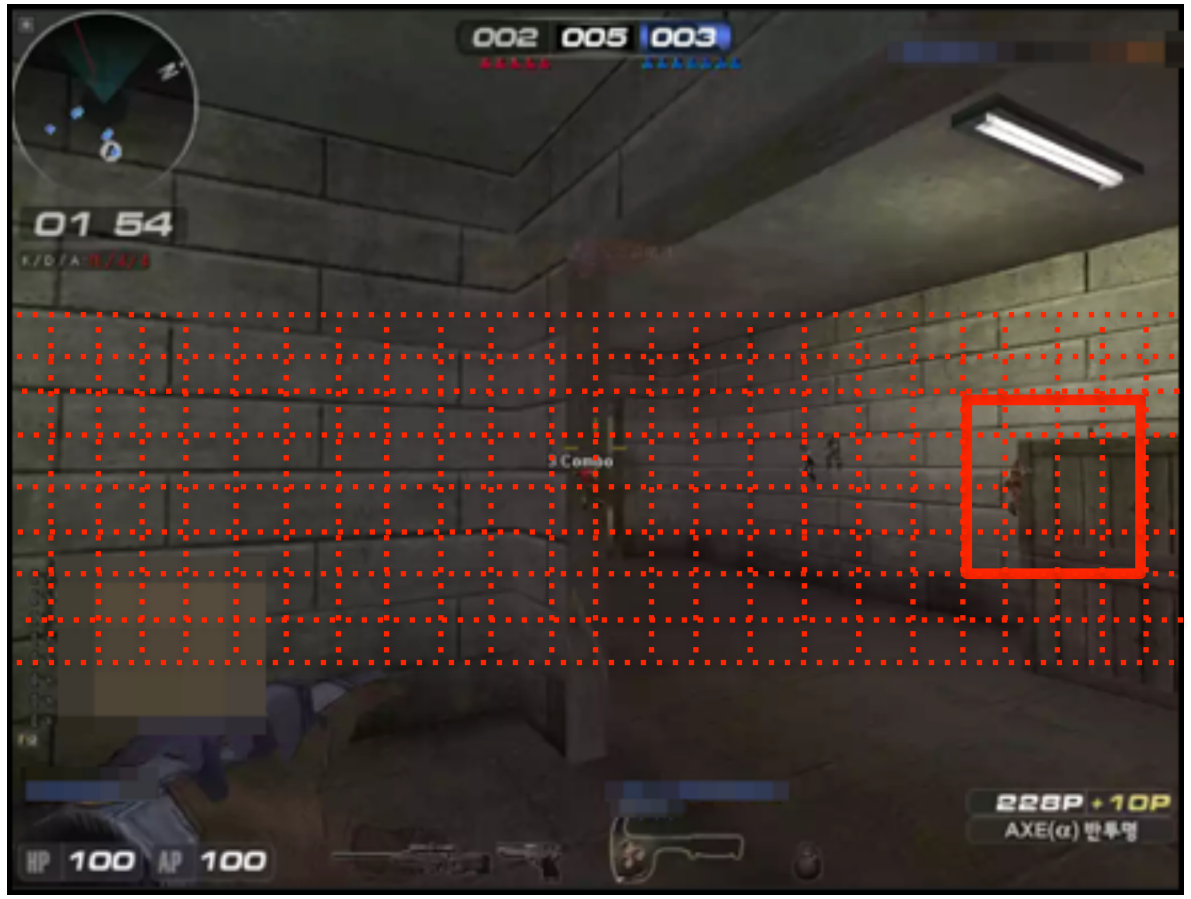
Unclear

Clear

Problem: Unclear Patches

- Fine-grained patches captures wallhack regions, but cropping and stitching becomes ***bulkier***



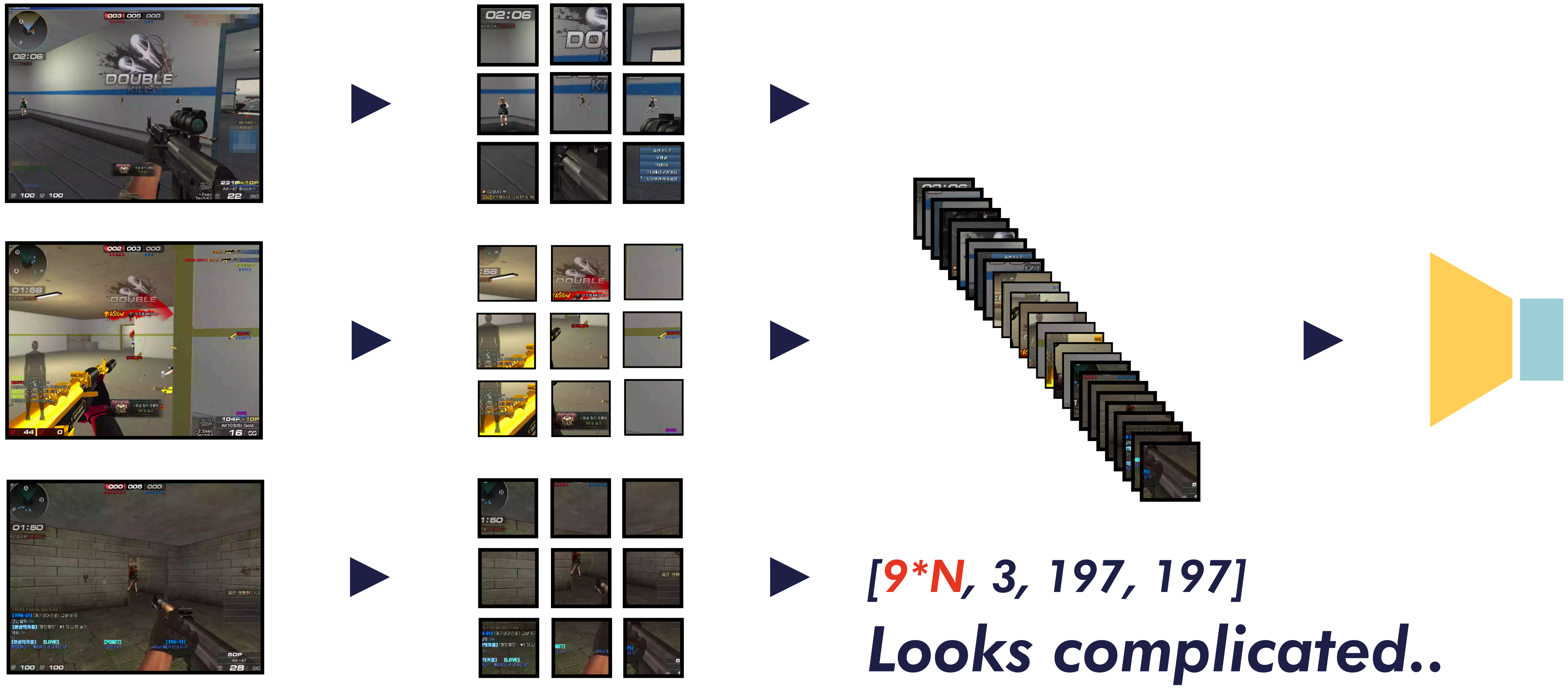


Problem: Inefficient inference

- Use ***batch*** dimension for multi-patch processing

Problem: Inefficient inference

- Can it process multiple screenshots in a ***single*** forward pass?

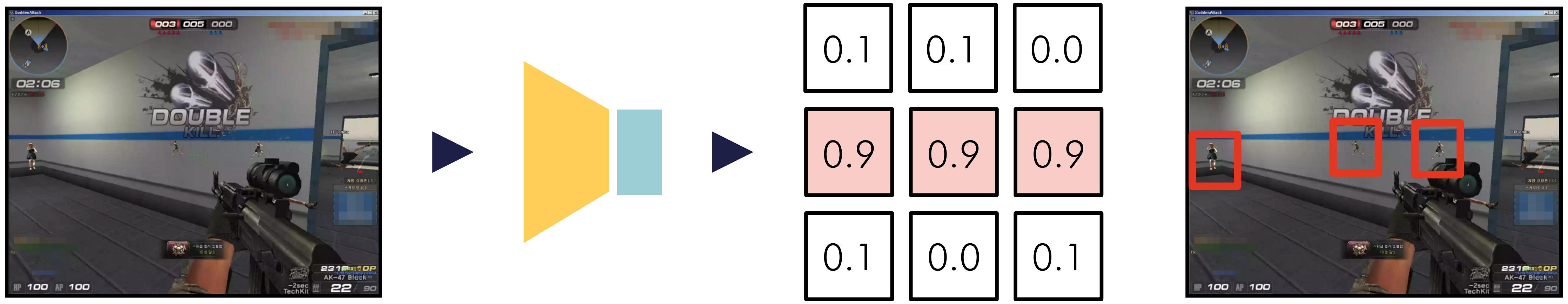


[9*N, 3, 197, 197]

Looks complicated..

More Convenient Inference

- Use patches to train the model
- And inference screenshots directly ***without cropping***



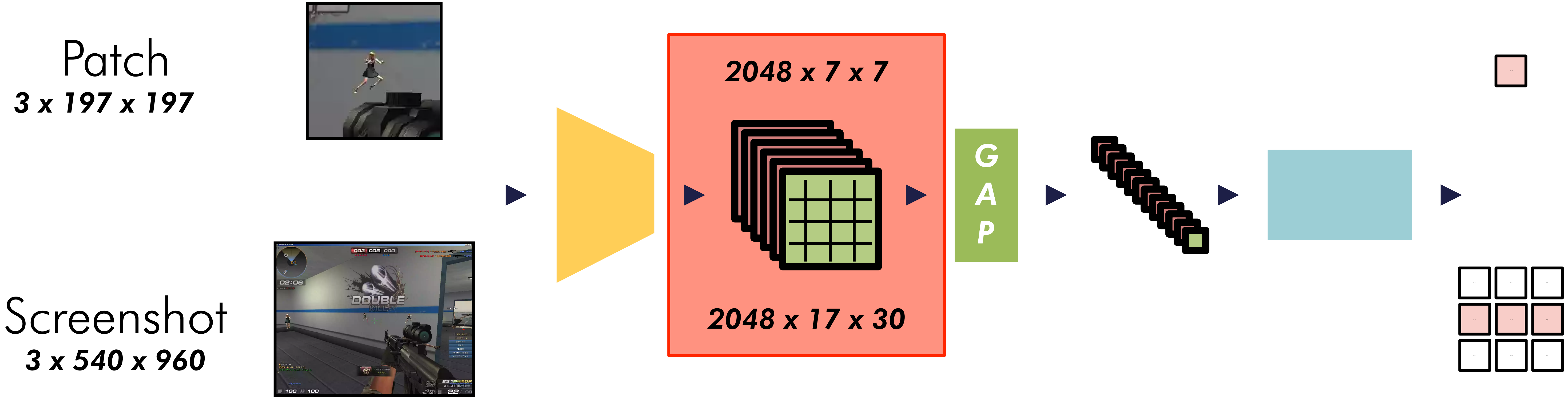
Original screenshot

Wallhack probabilities

Annotated screenshot

Screenshot as Input

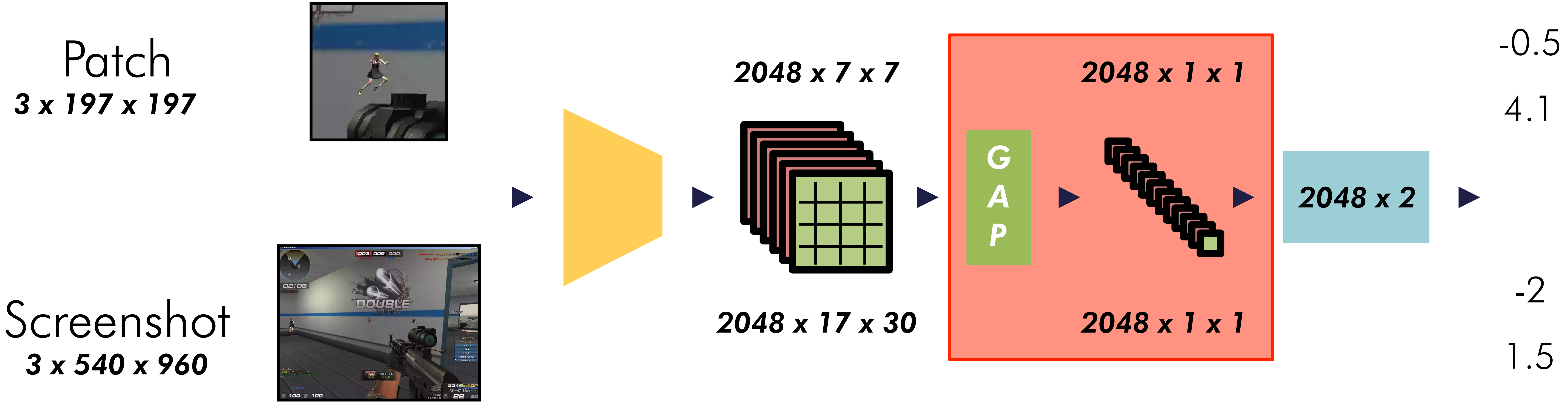
- ResNet50 **CAN** take screenshots without resizing thanks to Convolutional Layers



*batch dim is omitted

Screenshot as Input

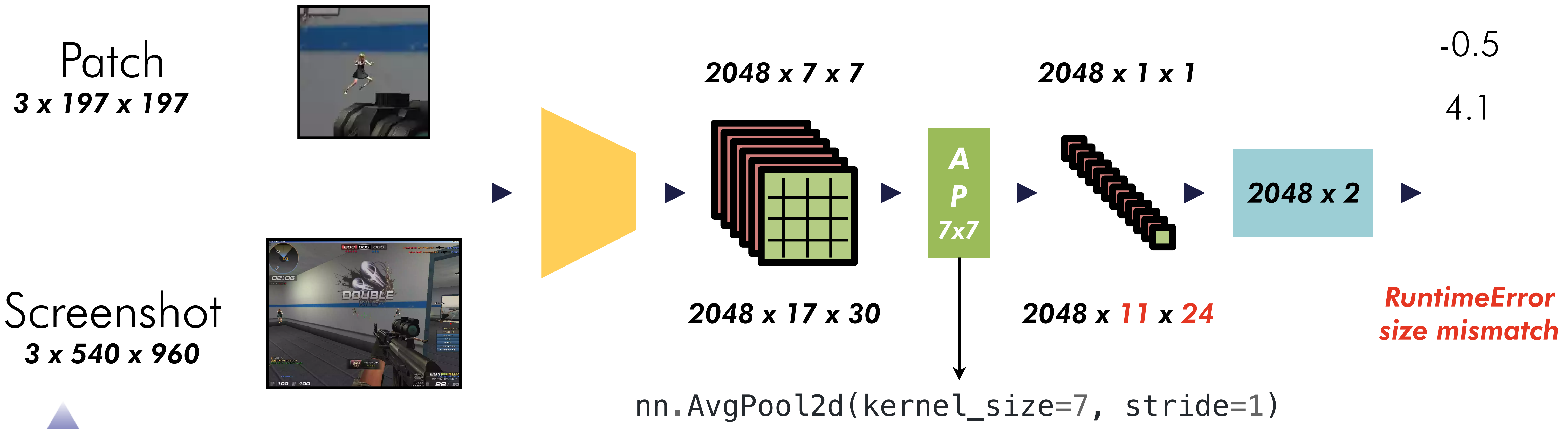
- ***GAP*** shrinks a feature map of any size into a single number



*batch dim is omitted

Problem: Can't process Screenshot

- AvgPool2d produces patch-wise information, but the linear layer returns ***RuntimeError***

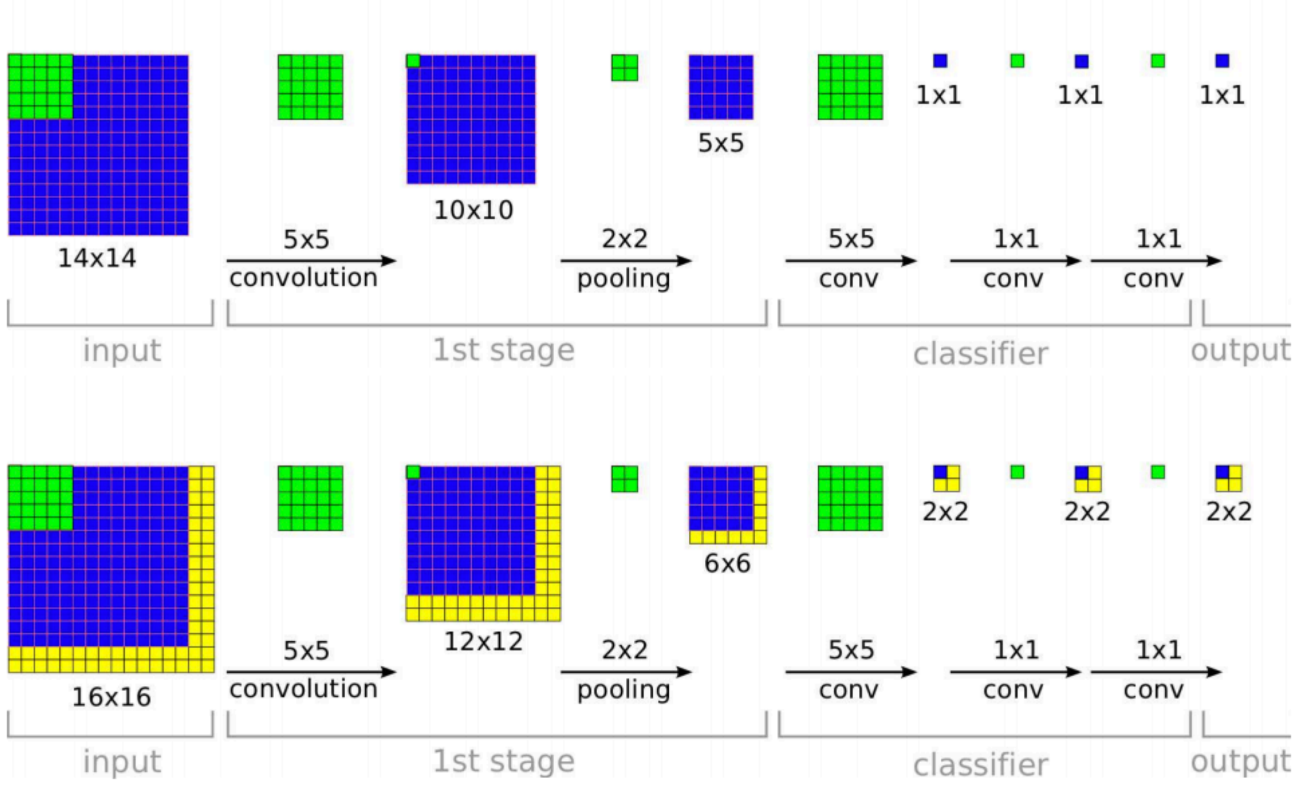


*batch dim is omitted

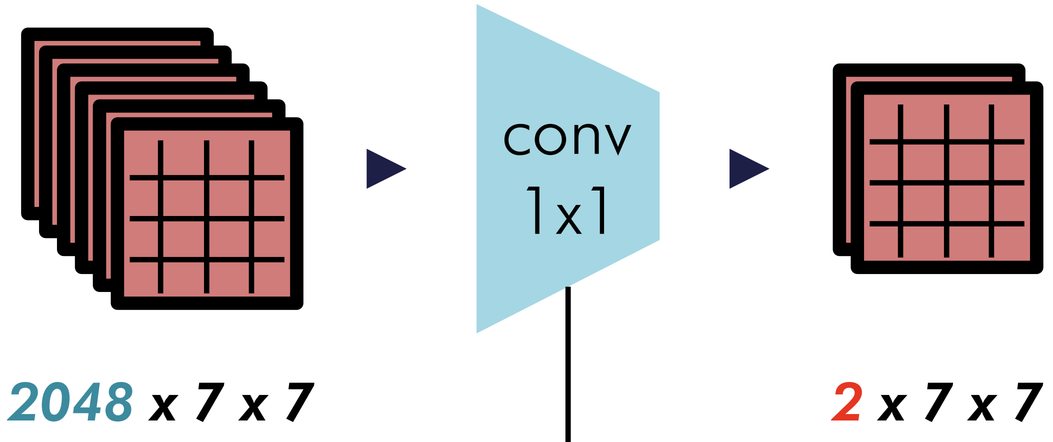
Idea: Fully Convolutional Network

- FCN's output is ***in proportion to*** the input size

"OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks" by Sermanet et al (2013)



1x1 conv manipulates channel dim




```
nn.Conv2d(2048, 2, 1, 1)
```

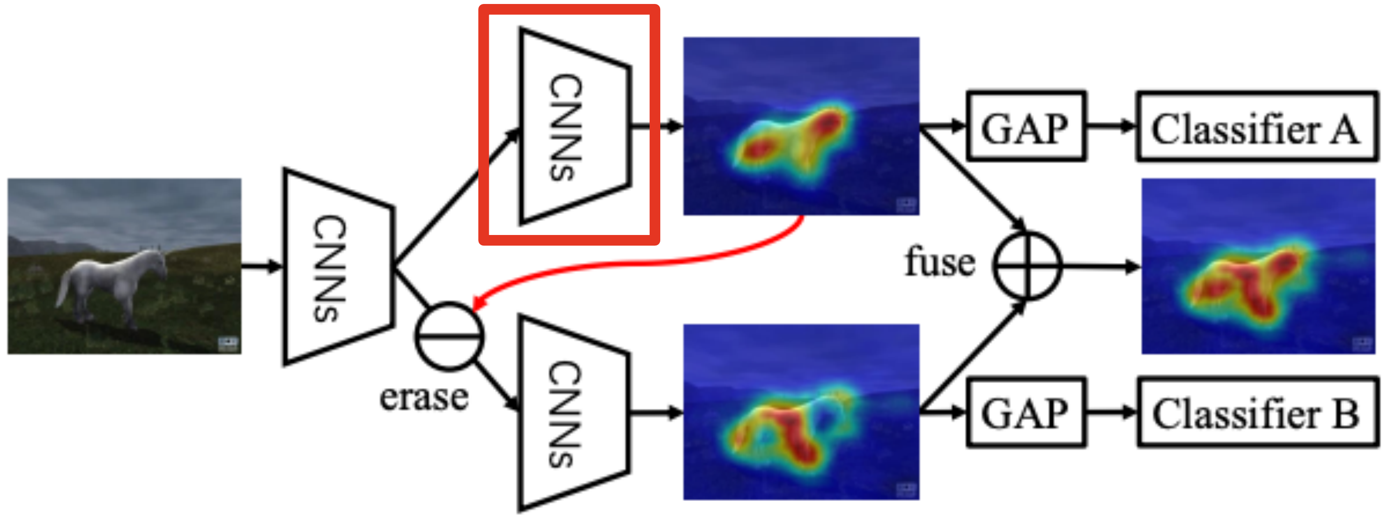
*batch dim is omitted

Idea: CAM with 1x1 conv

- Use ***1x1 conv***'s weight to generate CAM

"Adversarial Complementary Learning for Weakly Supervised Learning"
by Zhang et al (2018)

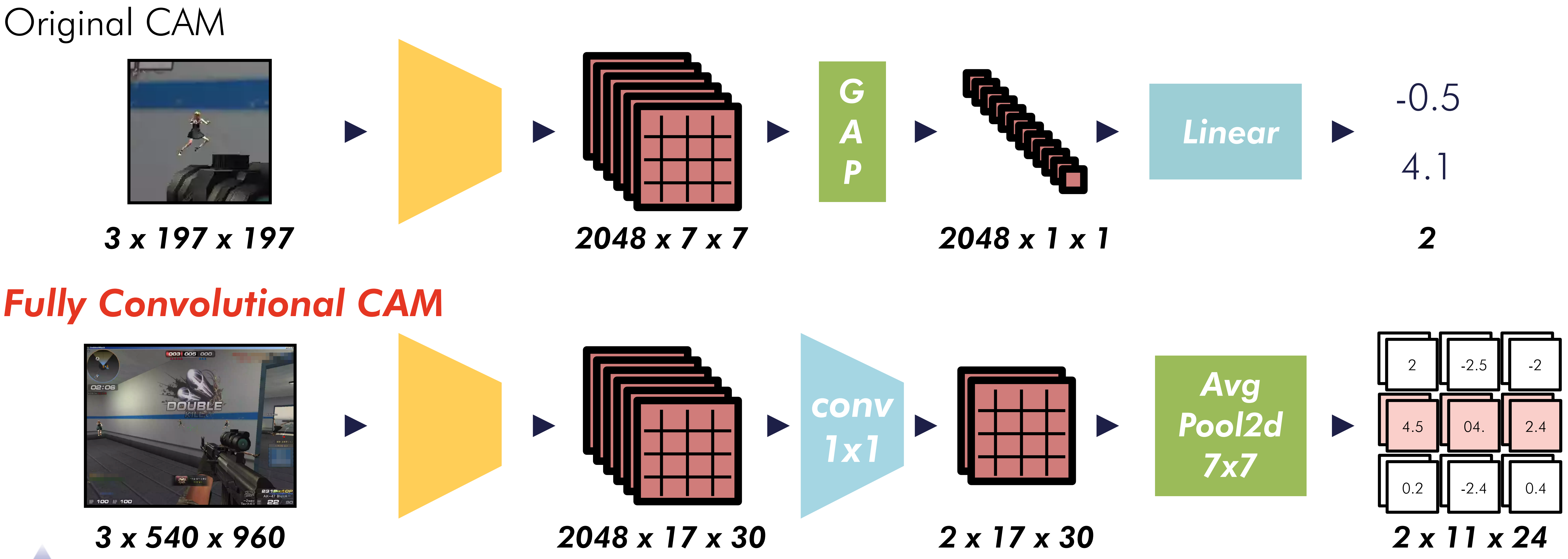
ACoL CAM
- Feature Maps
- ***1x1 conv***



Original CAM
- Feature Maps
- Linear Layer

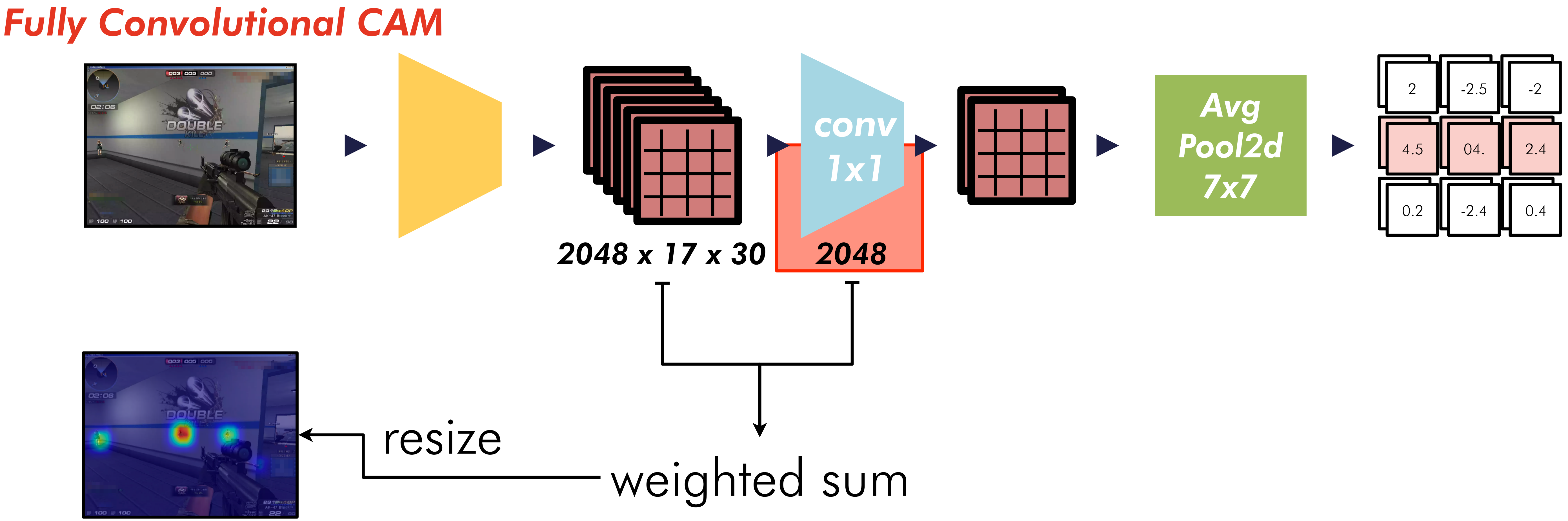
Solution: Fully Convolutional CAM

- ***AvgPool2d*** outputs a map of wallhack probabilities



Solution: Fully Convolutional CAM

- Use ***1x1 conv*** to generate CAM for screenshot



Effect: CAM without pre&post processing

Examples


001 005 001
01 47
1215
77P
Knife
100 100


000 005 001
02:07
1605 +205
100P +10P
Knife
100 100

Effect: debuggable dataset

- Use CAM to find ***helpful*** false positive patches

GDC

GAME DEVELOPERS CONFERENCE

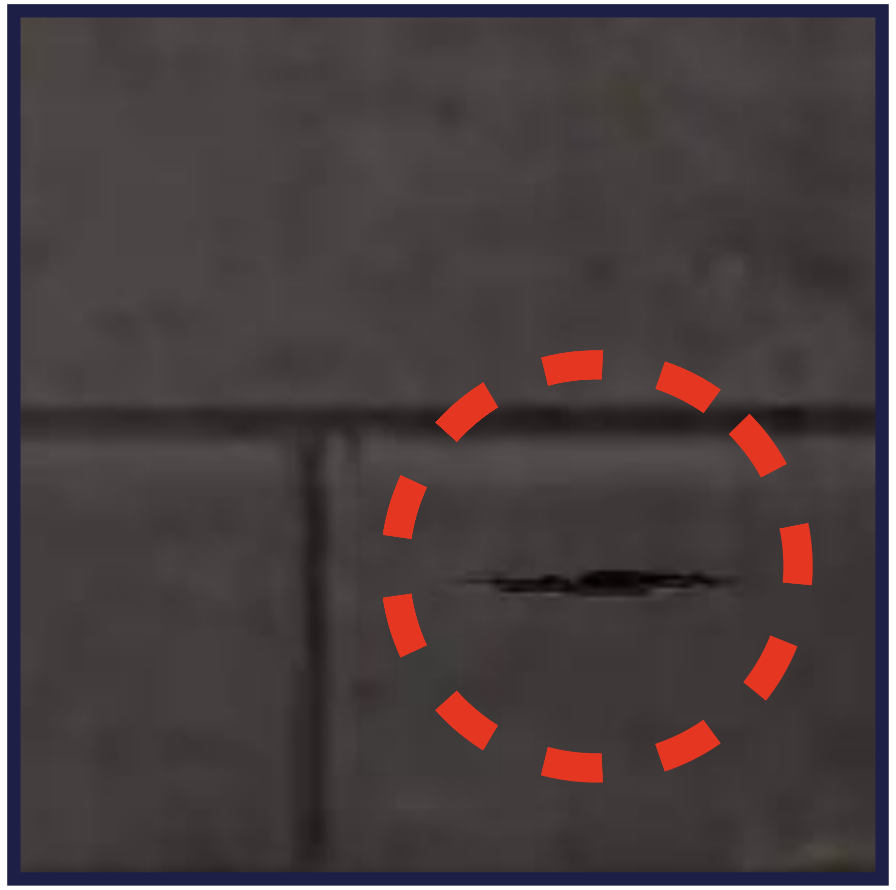
MARCH 18–22, 2019 | #GDC19

Effect: debuggable dataset

- Use CAM to find ***helpful*** false positive patches

Effect: debuggable dataset

- Use CAM to find ***helpful*** false positive patches



Effect: Active Learning with CAM

- Feed data that ***complement*** model's weakness

Passive Learning

```
for patch in unlabelled_patches:
```

Active Learning with CAM

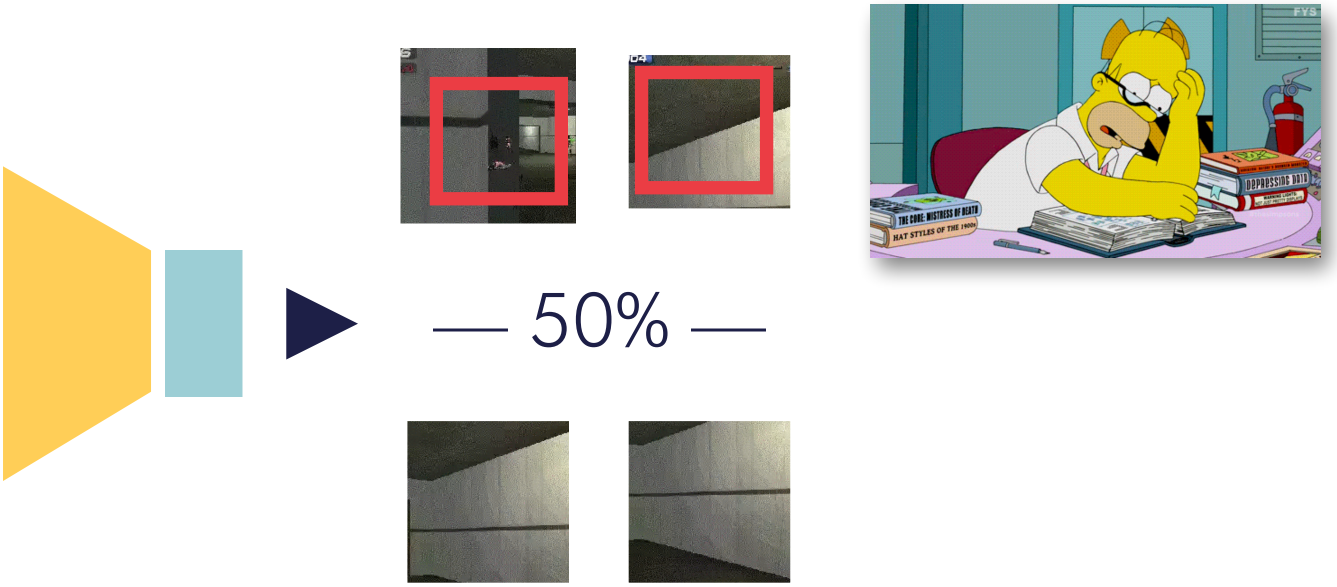
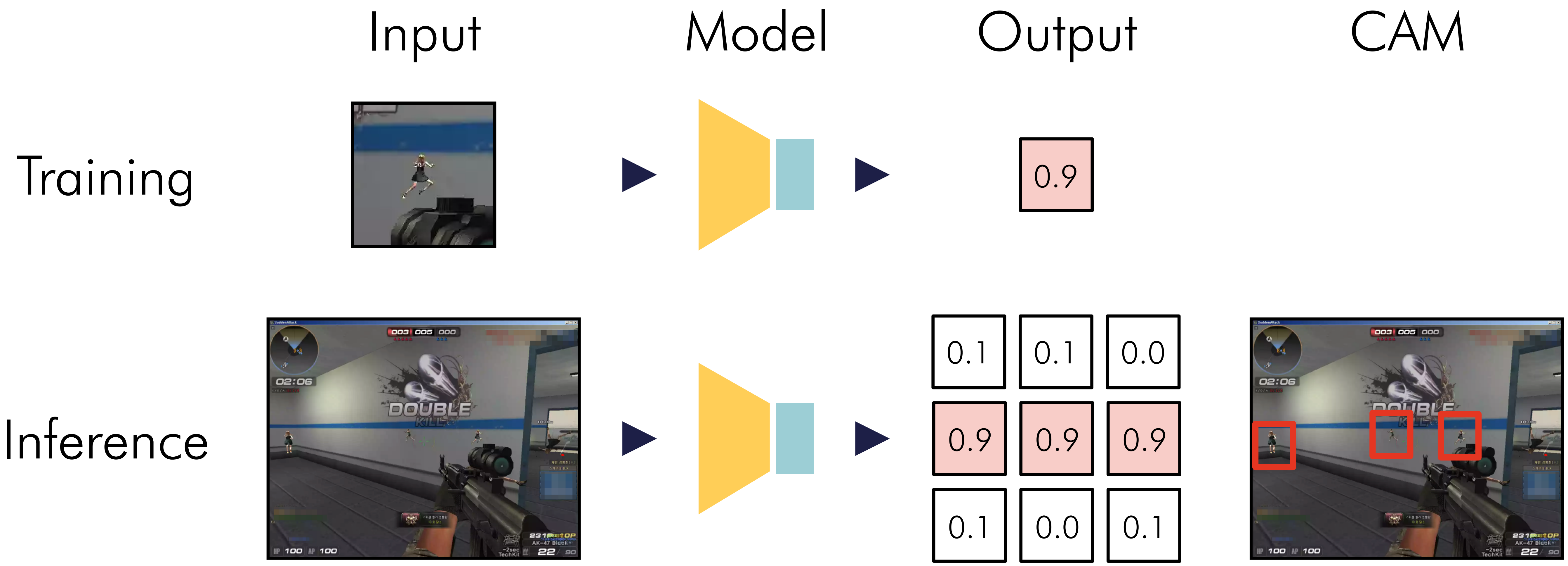


Image Credit: The Simpsons

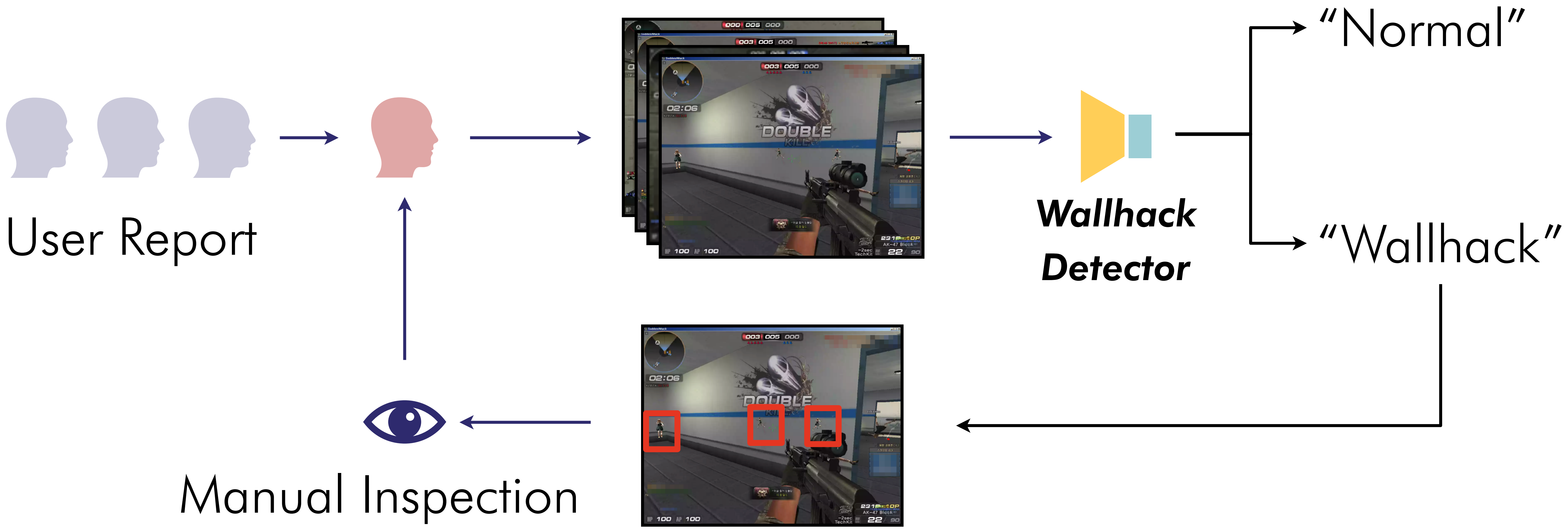
Project Output

Wallhack Detector

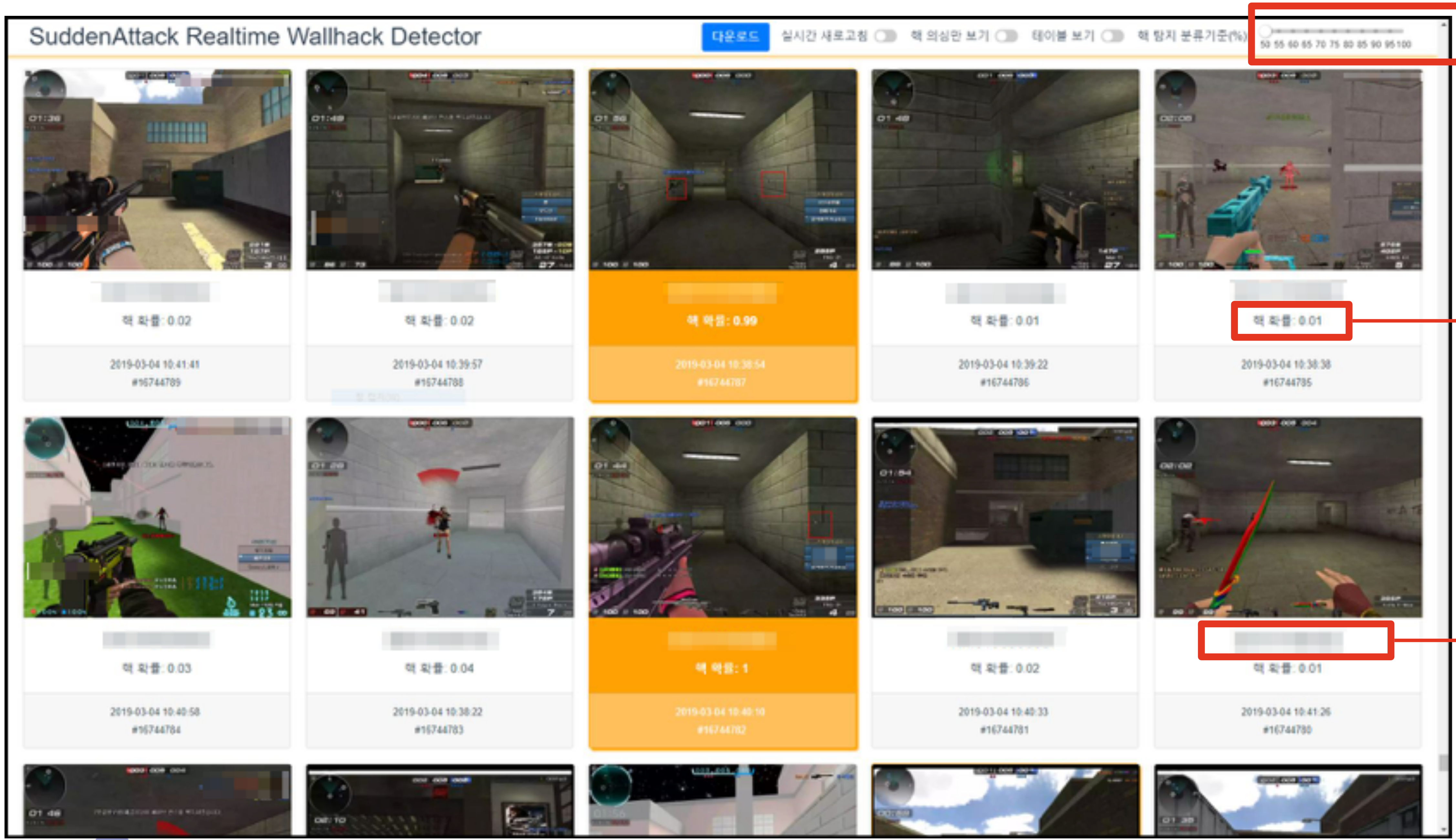


Wallhack Detector in Ban Process

- Filter out normal images and annotate wallhack regions



Realtime Dashboard



Probability filter

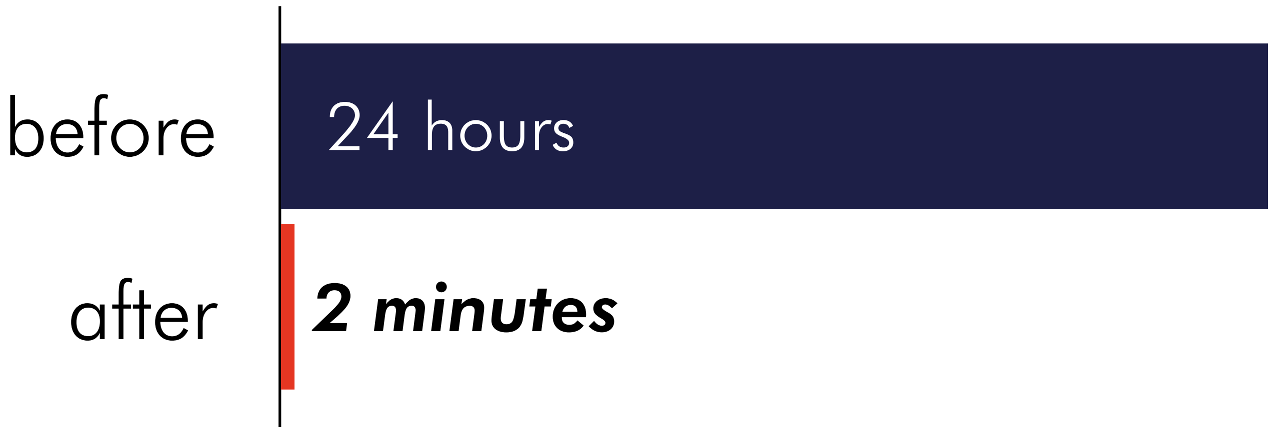
Wallhack probability

Account Number

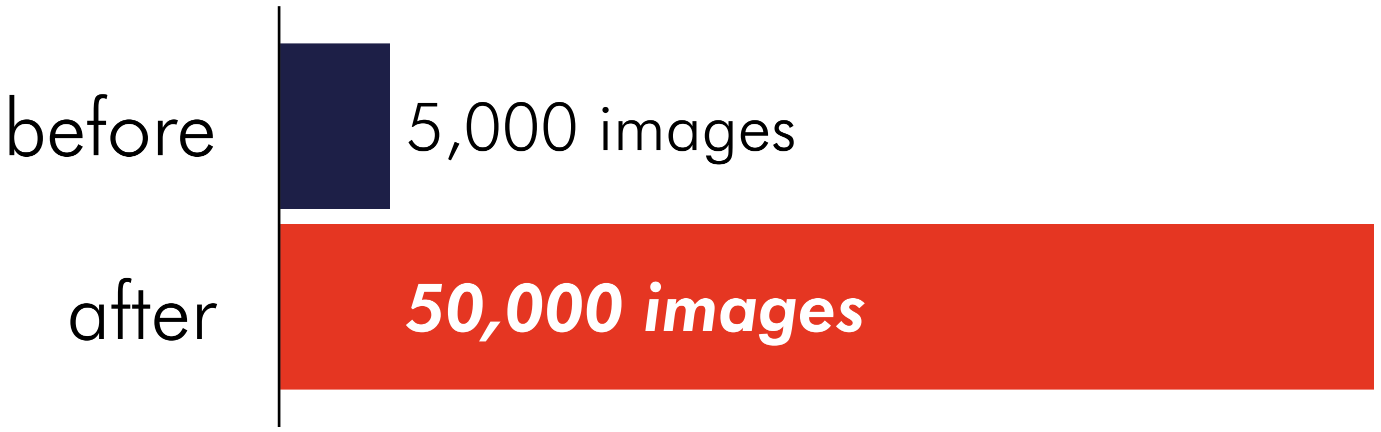
Project Output

- Abusers get banned in 24 hours ▶ ***2 minutes (0.001%)***
- Number of images to inspect: 5,000 ▶ ***50,000 (10x)***

Lead time until banned



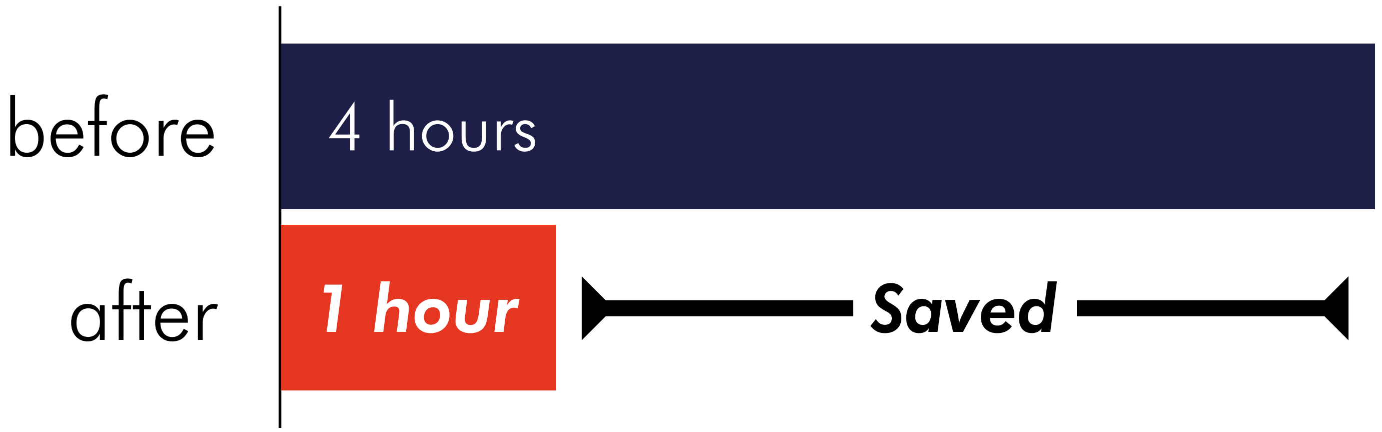
images processed daily



Project Output

- Shorten daily inspection: 4 hours ▶ ***1 hour (25%)***
- Free up community managers' time for more valuable tasks

Time spent for daily inspection



▶

Focus more on
- hack tool trades
- new hack types
- better gaming experiences

Takeaways

1. Leverage ***Transfer Learning*** when your dataset is small.

2. ***Handcraft features*** when the signal is too weak.

3. Use ***Class Activation Mapping*** to make NN interpretable

4. Go make your ***own***!

Questions?

Email – junsik.whang@nexon.co.kr

Blog – https://jsideas.net

https://career.nexon.com/

NEXON

References

[1] Stanford CS231n Transfer Learning

[2] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba. Learning Deep Features for Discriminative Localization https://arxiv.org/abs/1512.04150

[3] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. http://arxiv.org/abs/1312.6229.

[4] Xiaolin Zhang, Yunchao Wei, Jiashi Feng, Yi Yang, Thomas Huang. Adversarial Complementary Learning for Weakly Supervised Object Localization https://arxiv.org/abs/1804.06962v1

Appendix - CAMs

	method	Pros	Cons
Original CAM	linear weight based	- suitable for basic ResNet - forward pass	
Grad-CAM	gradient based	- compatible with any architecture - better results than CAM (sometimes)	- need gradient information - slower than CAM
CAM (ACoL)	1x1 conv weight based	- more convenient than CAM - forward pass	
LIME / RISE	occlusion based	- works on any algorithms	- takes too long in our case